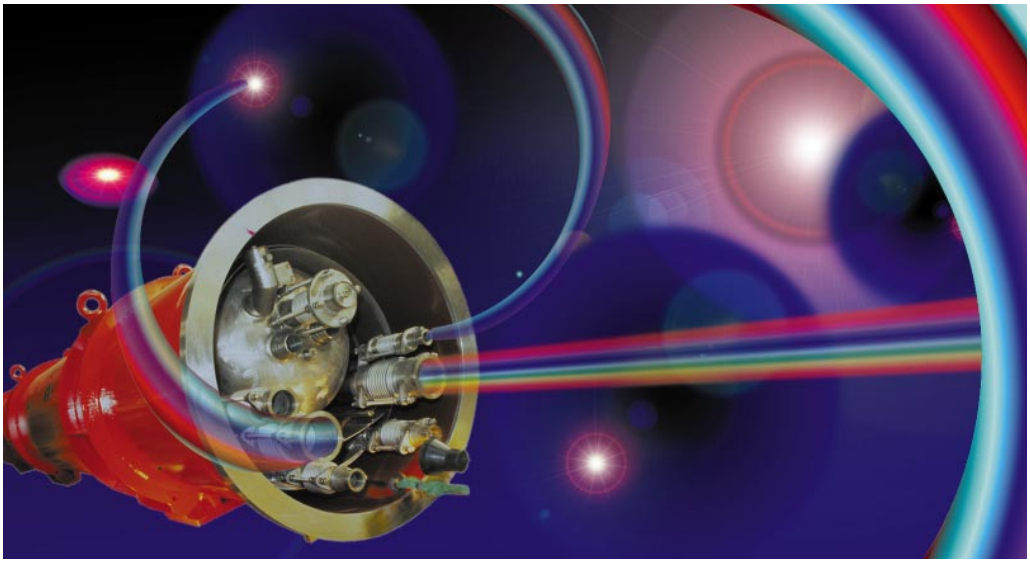
HOW TO STORE JAVA OBJECTS

U.S. $4.95 (Canada $6.95)

DEVELOPER'S JAVA™ JOURNAL

JavaDevelopersJournal.com

Volume: 3 Issue: 4



SYS-CON PUBLICATIONS

Full Page Ad

Full Page Ad

Full Page Ad

DEVELOPER'S JAVA JOURNAL

EDITORIAL ADVISORY BOARD

Ted Coombs, Bill Dunlap, Allan Hess,
Arthur van Hoff, Brian Maso, Miko Matsumura,
Kim Polese, Richard Soley, David Spenhoff

Art Director: Jim Morgan
Executive Editor: Scott Davison
Managing Editor: Gail S. Schultz
Editorial Assistant: Christy Wrightington
Copy Editor: Alix Lowenthal
Technical Editor: Bahadir Karuv
Visual J++ Editor: Ed Zebrowski
Visual Café Pro Editor: Alan Williamson
Product Review Editor: Jim Mathis
Games & Graphics Editor: Eric Ries
Tips & Techniques Editor: Brian Maso
Java Security Editor: Jay Heiser

WRITERS IN THIS ISSUE

Bob Adams, Daniel Dee, Joseph DiBella, David Gee,
Nancy Lee, Brian Maso, Sean Rhody, John Tabbone,
Joe S. Valley, Andrew Wade, 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 air-mail postage
(U.S. Banks or Money Orders). *Back Issues:* $12 each

Publisher, President and CEO: Fuat A. Kircaali
Vice President, Production: Jim Morgan
Vice President, Marketing: Carmen Gonzalez
Advertising Manager: Claudia Jung
Advertising Sales: Diane Baird
Paula Horowitz
Advertising Assistant: Erin O'Gorman
Accounting: Ignacio Arellano
Senior Designer: Robin Groves
Webmaster: Robert Diamond
Senior Web Designer: Corey Low
Customer Service: Rae Miranda
Sian O'Gorman
Mitchell Low

EDITORIAL OFFICES

SYS-CON Publications, Inc.
39 E. Central Ave., Pearl River, NY 10965
Telephone: 914 735-1900 Fax: 914 735-3922
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. Application to mail at Periodicals Postage rates is pending 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 © 1997 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.

ISSN # 1087-6944

DISTRIBUTED in USA by
International Periodical Distributors
674 Via De La Valle, Suite 204, Solana Beach, CA 92075 619 481-5928

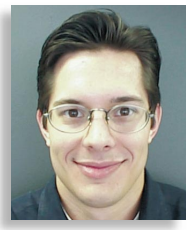
BPA Membership Applied For August, 1996

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.

SYS-CON PUBLICATIONS

GUEST EDITORIAL

Rhett Guthrie



Easy Does It

Visual Basic is arguably the most successful programming language in the history of programming languages. The number of VB components and applications out there is staggering, and the number of VB programmers is even more so. However, there is a not so silent contender for the World's Most Popular Language. It's OO, it's multithreaded, it's Internet-ready. It's an expressive and flexible language capable of industrial-strength server-side computing, and, for the C++ crowd, here's the real rub: it's idiot-proof. It's Java. Java not only promises enterprise solution-capable software, it promises to do so with VB-style ease of use and with an unrivaled adoption rate. Therein lies the central issue: The combination of ease of use, power and popularity makes Java an important language for the software engineering community.

Consider distributed computing. There has been an evolution of technologies including socket programming, RPC and distributed objects. For a while, however, we had distributed objects with a little baggage – you had to muck with your application logic and you had to generate stub and skeleton code. This model is rapidly being eclipsed in favor of easier and more seamless programming models. Clients of remote objects often need to know that the server object is remote to be prepared to deal with the inevitable gotchas of network computing. However, there is no justification for coupling the server object to the distribution layer – it need not know how it's being accessed. Therefore, some distributed computing platforms are removing this burden from the programmer and allowing any class to be remote-enabled without modification. In fact, even the tedious and error-prone proxy class generation step now happens on demand at runtime. It no longer requires intervention from the programmer. What does this mean? Distributed computing in Java is approaching the theoretical limits of ease of use!

The beauty of this is that it's just the beginning. Distributed computing is just one tool in a software engineer's bag of tricks. The entire gamut of software engineering is subject to this level of ease of use. Java's thread model has almost made platform-independent, multithreaded application development a non-issue. Garbage collection almost makes memory management a non-issue. Dynamic class loading greatly facilitates mobile agent platforms and applications. JavaBeans™ is quickly making GUI development a matter of connecting the dots. EJB promises to work similar magic for server-side transactional programming and persistence integration. JECF is on the way to solving the problem of developing electronic commerce software. The list goes on and on. Many of yesterday's programming nightmares are evaporating before our very eyes. As layer upon layer is added, we'll find more software development issues being taken care of automatically. Expect it and demand it.

Microsoft Windows NT 5.0 is supposedly comprised of no less than 25 million lines of code. That is significant by any standard. Now, imagine if the authors had to write this opus, not in C and C++, but in binary. Imagine the complexity of such a task. The number and quality of minds needed would be tremendous. You could make a strong case that human sociological development has not yet advanced to the stage that such a group project is possible. It would seem that an accomplishment like NT 5.0 is simply not feasible (perhaps not even possible) without the higher levels of abstraction provided by C, C++, COM and the rest. My own experience in building distributed computing technology provides at least a modicum of evidence that Java, with its flexibility, ease of use and power, has the potential to go even further. Java makes it possible to achieve instant distributed computing, and there is every indication that there is much more to come.

Let's step back and consider the big picture. Why do we even care that software is easy to build? Does it matter that Java technologies are easy to use? Definitely, because by making it easier for us to build software, we are improving our ability to solve problems. The human race advances by the number of operations it can perform without thinking about them. When Alfred North Whitehead said this, he probably wasn't thinking about software engineering abstractions, but his words couldn't be more applicable. Let's face it, software runs the world. Many improvements in the nature of medicine, government, quality of life, science and economy can be directly linked to improvements in software. By making it easier to build software, we're making it easier to advance as a people.

The last thing the Java community needs is more hype. It is certainly not my point to compound the hype problem. My contention is that Java and the emerging frameworks do, or at least can, facilitate software engineering better than the commercially viable alternatives on the market. I offer the ease and growing adoption of distributed computing in Java as a success story. Java isn't the solution for world hunger. It does, however, offer a compelling combination of popularity, ease of use and power. Software engineering is getting easier, and the Java platform is an important reason why.

About the Author

Rhett Guthrie is a Senior Technologist at Object-

Full Page Ad

SYS-CON PUBLICATIONS

PHONE, ADDRESS & WEB DIRECTORY

CALL FOR SUBSCRIPTIONS
1 800 513-7111

International Subscriptions & Customer Service Inquiries
914 735-1900
or by fax: 914 735-3922

E-Mail: Subscribe@SYS-CON.com
http://www.SYS-CON.com

MAIL All Subscription Orders or Customer Service Inquiries to

Java Developer's Journal
http://www.JavaDevelopersJournal.com

VRML Developer's Journal
VRMLJournal.com

National Java Learning Center, Inc.

JDJ Buyer's Guide
JavaBuyersGuide.com

Web-Pro Developer's Supplement

SYS-CON Publications, Inc.
39 E. Central Ave.
Pearl River, NY 10965 – USA

EDITORIAL OFFICES
Phone: 914 735-1900
Fax: 914 735-3922

ADVERTISING & SALES OFFICE
Phone: 914 735-0300
Fax: 914 735-7302

CUSTOMER SERVICE
Phone: 914 735-1900
Fax: 914 735-3922

DESIGN & PRODUCTION
Phone: 914 735-7300
Fax: 914 735-6547

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

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

David Gee

The Component Choice for e-business

Have you heard the words 'build virtual teams, extend the corporation, manage the supply chain'? Are you convinced that e-business, enterprise applications deployed over the Web, Internet plus intranet plus extranet are the way to go? Chances are you've thought about this and your answer is yes. But what does that mean to you, right now, as we're one Web year into 1998?

As you and your development team move from pilot projects to implementation of enterprise-wide systems across the Internet, intranet and extranet, you'll need to consider and balance several critical elements. The right combination of component architecture, client/server tools, Internet practices and existing legacy systems is crucial. In terms of decisions to be made this year, I would like to address choosing the right component model as one of the most important issues you'll contend with. According to a recent report by Forrester Research, Inc., of companies interviewed, 44 percent have no object strategy now, but by the year 2000 only 4 percent foresee having no object strategy in place.

Components, by definition, are self-contained program modules that can interact with each other. The software community is creating components to speed application development and to build up a platform-independent base of reusable code. By incorporating a well-conceived component model, you can anticipate, drive and respond to changing market conditions, optimize reuse and create custom applications more quickly.

Since you're holding this magazine in your hands, you've already decided that Java is an important element of the e-business equation, whether you're beginning to experiment with Java or are an advanced Java programmer. In Java, almost everything is an object or component. The JavaBeans™ spec was written by JavaSoft in conjunction with numerous industry leaders, including IBM. JavaBeans is fast emerging as the portable, platform-neutral component model written in Java.

The JavaBeans component architecture is the ideal choice for developing network-aware applications that allow you to move within the enterprise or across the Internet. Unlike days past, you can't assume central control over deployment. If you anticipate deploying systems over a heterogeneous environment, you'll want new systems to connect and integrate with any other hardware or software that might be encountered on the Internet, intranet or extranet. The JavaBeans component model places no restrictions on where applications can be deployed. And not coincidentally, JavaBeans connect into any other component model via bridges, including COM/DCOM. The opposite is not the case. The full COM environment – particularly Microsoft Transaction Server – will be available only on NT. Java and the JavaBeans architecture is the only model to consider with these goals in mind.

In 1997, we saw a lot of people building interesting client-side applications, just as before that we saw a plethora of spinning Java applets on the Web. The e-business equation rests on the belief that Java is not just a client-side model. Significant server-side Java initiatives are well under way, including IBM's massive San Francisco project. San Francisco has put over 300,000 lines of code in the hands of developers for creating run-your-business server applications. Over 250 companies have licensed the code so far, and the first of these applications will begin hitting the market later this year.

This server-side emphasis extends to the component model as well. The Enterprise JavaBeans spec is a component architecture for reusable server-side components to build business applications. IBM was a major contributor to this specification as well. Very soon we will begin to see support for scalable transactional application server components.

Other people in your organization who manage the desktops may be inclined to choose Microsoft's COM/DCOM model for your business. While COM/DCOM delivers decided benefits to the client, the real business benefit from component-based client/server lies in the business logic and applications that reside on the server. With leading visual development tools, Java's security model and built-in scalability, Enterprise JavaBeans is clearly the superior model.

Become the JavaBeans 'component proponent' within your company. You're going to be called on to make the quick changes and connections to the applications that run your e-business, so don't let the decision be made without you. Which component platform your organization adopts now will determine how and where you'll expand your business in the years ahead.

About the Author

As Program Director, alphaWorks & Java Marketing in IBM's Software Solutions, David Gee's role includes developing the company's Java marketing strategy, forging strategic business alliances and maintaining partner relationships with key industry influencers. You can learn more about IBM's Java initiatives at www.ibm.com/java and explore IBM's online research laboratory at www.alphaWorks.ibm.com.

How to Store & Share Your Java Objects

by Dr. Andrew E. Wade

The path you choose depends on your needs

Need to store your Java objects? Files can do this, with a little bit of programming to flatten them. Need to share them with others, guarantee integrity? Traditional DBMSs can do this, if you translate your Java objects to SQL. Need 24x7, scalability, distribution over WANs, flexibility for schema changes? ODBMSs can do this, and they can do it easily, by automatically making your Java objects persistent. We'll present the basics of object databases and contrast them with relational and object-relational; explain how to determine if your application is a good fit for ODBMSs; how to deal with legacy issues and how to use ODBMSs with Java and the Web. Examples, chosen from over 150,000 users in production, are included.

Where to Store Shared Information

Many software systems and applications deal with information which must outlive the process. There are many ways to save such persistent information. Broadly, these consist of file systems, traditional database management systems (DBMSs) and object DBMSs (ODBMSs).

File systems support the basic need of persistence. Information is still there after the process terminates and this information can be accessed later. Beyond this, files offer little and they do require work. The programmer must somehow flatten his Java objects into streams of primitive data, and then manually write those streams to files. The reverse is necessary to access the information later. Any changes in the object types will likely require changes in this flattening code, in the file format and perhaps in applications that use it. Any concurrent access control is up to the application programmers or conflicts will result. Files are useful when you have a small amount of information, which is unlikely to change much, accessed by only a single user (at a time), with no need for reliability features such as recovery or usability features such as relationships, distribution and versioning.

The next choice for persistence is to use a traditional DBMS, such as a relational one (RDBMS). These systems have been very successful in business applications which use very simple, primitive, fixed-length data types, organized in tables. They add support for concurrency so multiple users can access the same information without destroying each other's work. They also add recovery, so the stored information can be restored to a known, good state even after power outage or other catastrophic failures. They add powerful searching (or query) capabilities. Unfortunately, RDBMSs were designed for a different generation of software technology in which users dealt with raw (unencapsulated) data, third generation programming languages (COBOL, FORTRAN, C) and a data-specific language (SQL), with the programmer manually translating back and forth between the two. With objects, this means the programmer must translate his objects to flat, primitive types and sort them by tables. Then, when restoring the objects from the RDBMS, the programmer must reassemble the objects from various tables, using slow inter-table connections called joins. This mapping code results in three problems:

- **Programmer Time:** Instead of writing mapping code, programmers could be developing (and maintaining) more applications. It is not uncommon to see a third of an object application dedicated to this mapping code.
- **Integrity:** Because the RDBMS deals only with the low-level primitives, all higher-level application object support, including methods that maintain their semantics and integrity, are unknown to the RDBMS. Instead, applications must enforce such integrity constraints by translating the data into objects and using the object methods. If different applications do this differently they will be out of synch, causing integrity violations. End user graphical tools (forms, reports, query tools) go directly to this primitive level and thereby bypass any of the object constraints, losing integrity.
- **Performance:** At runtime, the need to disassemble and reassemble objects takes substantial time, slowing the application.

	File	Traditional DBMS	ODBMS
Persistence	v	v	v
Recovery		v	v
Concurrency		v	v
Objects			v
Integrity			v
Distribution			v
Scalability			v
Relationships			v
Versions			v

Table 1: Comparison of file systems, traditional relational databases and object databases

ODBMSs include the capabilities of traditional databases, but add several new ones. First, they support objects. The very same objects you define and create in Java are transparently managed by the ODBMS, including saving them on disk, recovering from failures and coordinating concurrent access. This means there is no need for the mapping code (described previously) with all of its problems. It also means that all the DBMS capabilities, including recovery, concurrency, and query with object methods, apply directly to objects rather than to the primitive, disassembled pieces of objects. Because all access to the ODBMS goes to the objects themselves, they can automatically enforce integrity. Even graphical ODBC tools can be forced (using security restrictions, by user and group) to go through high-level object methods in order to maintain integrity and most ODBMSs.

Where RDBMSs have mainframe-like central-server architectures in which all storage and processing occurs on a centralized machine, certain ODBMSs have been developed with distributed architecture. This allows objects to live on any computer (accessible in the networked environment), to execute anywhere and to be accessed transparently by all users, with all operations working across this distributed single logical view. In addition, objects make a natural unit for replication, and implementations now exist that keep replicas in synch even across failures. The distributed ability to support transparently adding servers is a major part of scalability, and other capabilities have been added, too, including concurrency modes that support multiple readers and up to one writer running simultaneously without blocking.

Also, ODBMSs bring new features. The ability to define many-to-many relationships allows the ODBMS to generate and

Front End	R	O	O	R
Front End	R	O	R	O
Tech	RDBMS	ODBMS	ORDBMS	ODBMS + RDBMS
Example	Oracle	Objectivity/DB	Illustra UniSQL	Objectivity/SQL++ PowerBuilder,...
Fits	Simple Data OLTP	Complex Info Relationships Distribution	Simple OO Tools	Complex + Query & Tools

Table 2: DBMS architectures: ODB vs. RDB vs. ORDB

manage the code to maintain such relationships, to dynamically add and remove elements and to maintain referential integrity, all without the need for users to write code or manually manage secondary data structures such as foreign keys. Moreover, traversal of such relationships is direct, without the need to search down tables and compare keys as is done in the relational join. By connecting networks of objects with these relationships, users can construct composite objects, which allow any number of levels of depth, and also any number of composites threading through a single object. Objects also provide the natural unit for versioning, keeping track of the history of the object's state, or even allowing simultaneous creation of multiple branches. Finally, since these features are in the DBMS, rather than layered over some simple data-only store, the ODBMS can integrate them together to properly handle complex object's models; e.g., recovery of composites and relationships and the behavior of relationships when one of the objects versions, etc.

In brief, ODBMS brings the advantages of files and traditional DBMSs, and also adds support for objects and additional features.

What about ORDBMS?

Faced with customer requests for object support, the RDBMS vendors have come up with an approach called object-relational or ORDBMS. To understand this mixed approach, we'll look at the high-level architectural description shown in Table 2.

A DBMS architecture can be split into the front end, which interfaces to the user, and the back end, which stores and retrieves the persistent information. *Either* of these may be based on *either* relational or object technology, providing the four alternatives shown. The first, with relational front and back ends, gives a typical RDBMS, while the second does the same for ODBMS. The third shows an object front end layered over a relational back end engine. This is the approach of the RDBMS vendors, largely because they have a large investment in their back ends and it's very hard to change them. Adding the object front end does add value; e.g., it might allow better integration with some object tools and it might allow some new data types. However, the back end is still relational, which means the objects are still being disassembled into flat tables, or BLOBs, whose internal structure is unknown to the rest of the DBMS. Some ORDBMSs are adding data "blades" or "cartridges" which are effectively pre-built class libraries. Unfortunately, they miss the point of objects by dealing only with data. Also, they require kernel modifications, so they are hard for typical users to build, or even modify. In contrast, ODBMSs allow users to freely build any classes of objects, with any operations and relationships and to freely extend others' classes. All of these can be used in exactly the same ways as any pre-built classes.

For completeness, the last column of Table 2 shows how a relational technology front end (including query and ODBC) can be layered on top of an object database back end. This not only adds functionality, including ad hoc query of objects and off-the-shelf use of all the familiar tools, but also plays a key role in legacy support, as we'll see below.

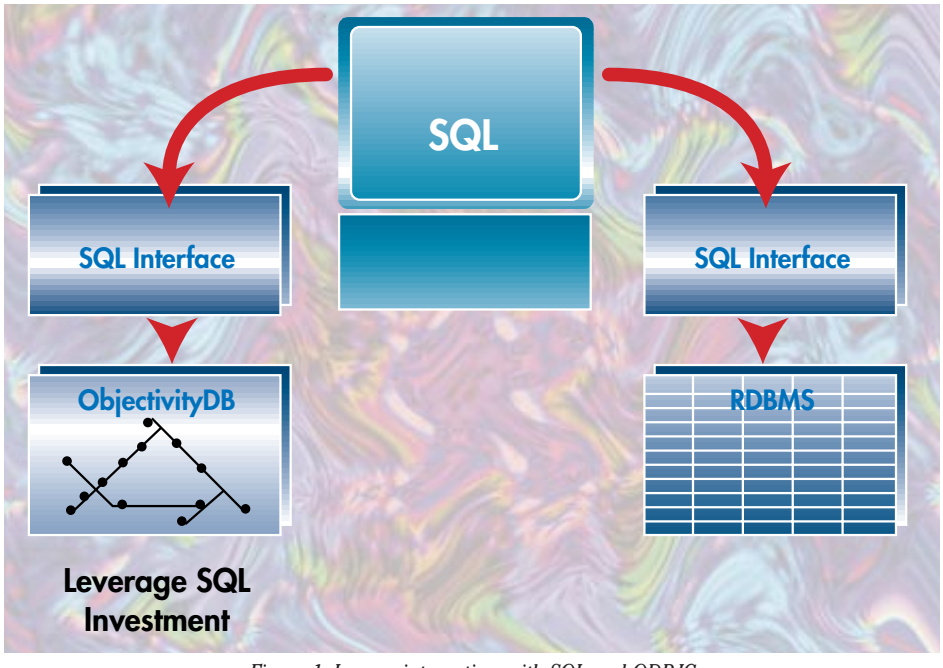


Figure 1: Legacy integration with SQL and ODBJC

When to Use an ODBMS

If your information needs to include any of the following, a DBMS is likely to help:

- Recovery
- Concurrency
- Integrity Management
- Scalability
- Security

An ODBMS may well be a better tool for maintaining your persistent information if any of the following three items apply to your system:

Object Usage

If your application or system is designed and built using objects, that in itself might make an ODBMS a better choice. It means the same Java objects are directly managed

Full Page Ad

by the ODBMS. There's no need to translate them to and from some other format (tables, records, etc.). This makes it easier to build and modify the system, faster to execute, of higher integrity and more likely that the system will come together correctly because the same modeling abstractions are used throughout.

Complex, Interconnected Information

While RDBMSs work well for flat, fixed-length primitive data, sorted by type into tables, many applications require more. In fact, most applications never used traditional DBMSs, and newer applications are using yet more complex information, including variable-sized structures (e.g., time series data), nested structures, images, audio, video and whatever someone might dream up tomorrow. All these are modeled directly as objects, making them

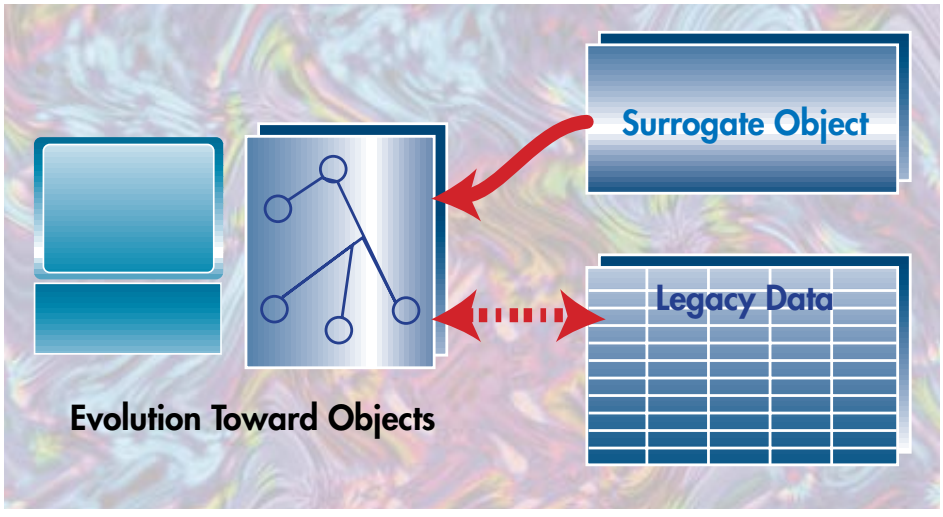


Figure 2: Surrogate objects integrate legacy systems

easier and faster to use. Even more important for some users are the relationships. RDBMSs have no direct support for these, requiring users to create secondary data structures (foreign keys) and manage them directly. Worse, the RDBMS uses a slow, non-scaling, search-and-compare process (join) to determine what is connected to what at runtime. The direct ODBMS support is much easier, faster and includes more capabilities. A common rule of thumb is: If you have more than three or four joins, it's worth looking into an ODBMS.

Distributed Environment

Traditional, relational and even some object DBMSs are built around mainframe-like central-server architectures. For some applications, this works well. For others (more and more these days), the deployment environment consists of multiple servers, workstations and PCs, often with more computing power scattered around the network and desktops than is contained in the central computer. The users wish to store objects anywhere, access them from anywhere, execute them anywhere, including multiple tiers. An ODBMS can do this much better with a distributed architecture, using objects as the natural unit for distribution and object identity as the basis for transparently locating objects. This can work over networks of heterogeneous computer hardware, operating systems, networks and compilers. Even separate languages (C, C++, Java, Smalltalk, SQL/ODBC) can be used to simultaneously share, access and modify objects, a key capability for object technology because it enables reuse.

Users

The earliest users of ODBMSs were those who had no choice, because they simply couldn't use the traditional DBMSs, yet they still had a significant need for persistence of large amounts of information, concurrency, scaling and recovery. These were engineering applications such as CAD/CAE, both mechanical and electronic, and are still users. Scientific applications also are major users. Examples here include CERN, in Geneva, storing the results of high-energy physics experiments (pictured on pages 8 and 9). They're building the world's largest database, 100 PB (a petabyte = 1,000 terabytes = 1,000,000 gigabytes). Similarly, the Sloan Digital Sky Survey (FermiLab, Johns Hopkins, etc.) is building a 40TB database containing the first digital survey of the sky, storing the stars, galaxies, quasars, etc., as objects in the ODBMS.

From there, the user base expanded into Telecommunications, where network management and real-time call routing require the performance, direct relationships, scalability and flexibility of ODBMSs. Examples here include Qualcomm (and their customers Nortel, Sprint, etc.), creators of the CDMA cellular standard, who build all their base stations on an ODBMS. Other examples include Siemens' Multiplexor, Intecom's Voice/Video/Data PBX, COM21's cable TV-based very high-speed modems (up to 1Mbps) and Motorola's Iridium satellite-based world-wide cellular system.

Manufacturing and process control are another major user, with real-time support for controlling distributed environments as well as databases of historical information for off-line analysis and query. Users in this area include Fisher-Rosemount, manufacturing control systems widely used in the petroleum and chemical and pharmaceutical industries; Landis & Gyr, environmental control systems used to maintain the world's busiest airport, Chicago's O'Hare; the Transamerica Pyramid and hospital suites, etc.; and KLA-Tencor, the market leader in semiconductor manufacturing.

Financial services are just now becoming users of ODBMSs, as exemplified by Citibank's currency trading system, deployed across Europe and the USA. Logistics systems such as BBN's Target are used in military and commercial environments, as are transportation systems. Others include document management, library management, healthcare systems, plus the utilities industry where American Meter has built a data collection application for remote meter reading and demand-side management.

The ODMG Java Interface

If you've looked at DBMSs before, you may be surprised to see what it looks like to use an ODBMS. Unlike traditional DBMSs, the ODBMS approach is to integrate the DBMS functionality directly into the host language. For Java, this means you simply define, create and access Java objects normally and the DBMS takes it from there. Of course, there are places where you will want to explicitly use the DBMS; for example, to start and end transactions (for recovery points and points where your work becomes visible to others), to create and access large collections, many-to-many relationships, etc.

The Object Database Management Group (ODMG), a consortium of vendors and users of ODBMSs representing essentially the entire vendor community, has defined standard interfaces to ODBMSs. You can read about their latest work at http://www.odmg.org/ or in the book, *"The Object Database Standard, ODMG 2.0,"* from Morgan Kaufman. The Java binding works with ODMG's Object Definition Language (ODL), and thereby OMG's Interface Definition Language (IDL), as well as ODMG's Object Interchange Format (OIF) and Object Query Language (OQL), which is quite close to, but not exactly like, the SQL2 query (SELECT-FROM-WHERE).

The normal syntax is used within Java to

Full Page Ad

define object types, instantiate objects and access them. Persistence is via reachability, which means that once an object is connected to a persistent object (including "root" objects), it becomes persistent. This is a natural extension for dynamic, garbage-collected languages in which unconnected objects are considered garbage and (eventually) deleted. Objects connected to other transient objects are retained transiently (until the end of the process), while those connected to persistent objects are retained persistently (across processes, until they become garbage). A brief example is shown in Listing 1.

Legacy System Access

It is a rare designer who has no legacy system to deal with. Luckily, ODBMSs provide a couple of very good ways to link new, object systems to older, non-object legacy systems. The two most common approaches are first, based on SQL, and second, based on surrogate objects – both of which can be used if desired.

Since some ODBMSs now fully support SQL and ODBC, these well-known languages may be used to simultaneously access both the objects in the ODBMS and the tables in legacy RDBMSs. Programs written in SQL can access all such systems, as can the familiar graphical tools (Crystal Reports, Microsoft Access and Visual Basic, etc.), almost all of which support ODBC (see Figure 1). The advantage of this approach is that it leverages existing investments in programs, tools and also in personnel training. Experienced database users can immediately access the new (as well as old) databases, starting where they're already familiar, and over time learn more and more about objects in order to get more benefits.

For the object user, a preferable approach would be to make the legacy systems accessible as objects. This is done by creating surrogate objects, which stand for information in legacy systems. For the major RDBMSs, class libraries to do this can be purchased; for these or other systems, the user can also write his own surrogate methods to read and write legacy information. The result is that these surrogates fit transparently into the distributed, single logical view. When they're accessed, they go off to the legacy systems but, except for performance considerations, they look exactly like any other objects. Although the mapping of tables to objects can be done automatically in a straightforward way, it is usually best to reanalyze the entire system, define the desired view of objects and then bury in the surrogate's methods the translation to any historical structures, so objects might be pieced out of different tables or go through legacy modules as needed to meet the application's and user's functionality. The result is that the new object users have full access to the legacy systems, but the legacy systems themselves continue to work unchanged. Evolution is now possible at the user's discretion and timetable: legacy information can be moved into native objects if and when desired, with no change for object users though of course at that point legacy systems will need to be changed to use the native objects (see Figure 2).

"Unlike traditional DBMSs, the ODBMS approach is to integrate the DBMS functionality directly into the host language."

Conclusion

The path you choose depends on your needs. For batch storing/restoring of a small number of objects, with little concern over speed, flattened streams to flat files work. For concurrency, recovery, backup, etc., go to a DBMS. The most natural, easiest and most efficient DBMS approach is ODBMS, which also can add native Java binding (just code Java and the ODBMS works underneath), performance, scalability, reduced programming cost, extra integrity, relationships, versioning, composites, kernel-level support for extensibility. Some ODBMSs can also add 24x7 support (online administration, garbage collection, schema evolution, etc.), fault tolerance, replication, transparent distribution, heterogeneity (simultaneous use of mixtures of different operating systems, languages, applications and databases).

Finally, unless you like the "bleeding edge," check for references that are successful in production, using the features or capabilities you need.

About the Author

Dr. Andrew E. Wade is the Founder and Vice President of Objectivity, Inc. He helped found both the Object Management Gourp and the Object Database Management Gorup and has co-authored and contributed to several books and written many articles. Objectivity can be found at www.objectivity.com. Drew can be reached at drew.wade@objectivity.com

drew.wade@objectivity.com

Listing 1: Examples of Java use of ODBMS.

```
//a persistent class with a transient attribute
public class Person {
public String name;
transient Something currentSomething;
...}

// Opening a database
public static Database open(String name, int accessMode)
throws ODMGException;
public void close() throws ODMGException;

//standard java code applies for accessing objects

//example code using collections and OQL queries
SetOfObject mathematicians;
mathematicians = Students.query(
"exists s in this.takes: s.section_of.name = \"math\" ");

Bag mathematicians;
Bag assistedProfs;
Double x;
OQLQuery query;
mathematicians = Students.query(
"exists s in this.takes: s.sectionOf.name = \"math\" ");
query = new OQLQuery(
"select t.assists.taughtBy from t in TA where t.salary > $1 and t
in $2 ");
x = new Double(50000.0);
query.bind(x); query.bind(mathematicians);
assistedProfs = (Bag) query.execute();
```

Full pg

ADDING A MIDDLE TIER TO YOUR JAVA CODE USING JAGUAR CTS

Making the power of Open Server usable for the component builder, regardless of language

by **Sean Rhody**

The saying goes, "Don't put all your eggs in one basket." In the programming world, particularly in larger corporations, this is tantamount to a doctrine of the faith. The client/server revolution introduced the idea of database independence from application logic. The birth of the Web and the desire to reuse business logic have also led to the development of a third layer – known as the Middle Tier.

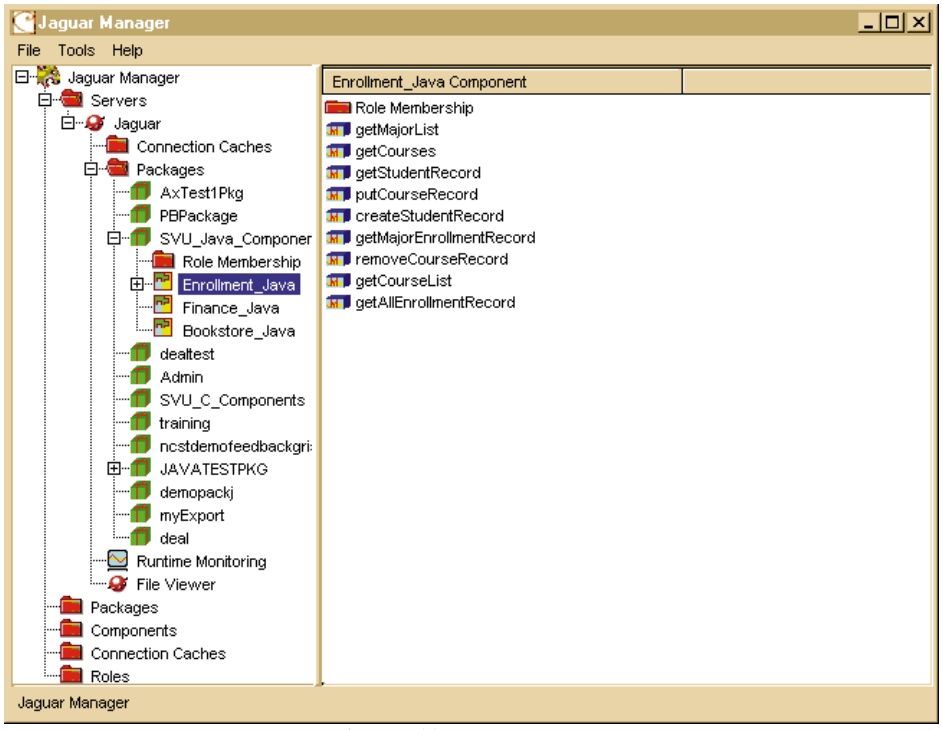


Figure 1: The Jaguar Manager

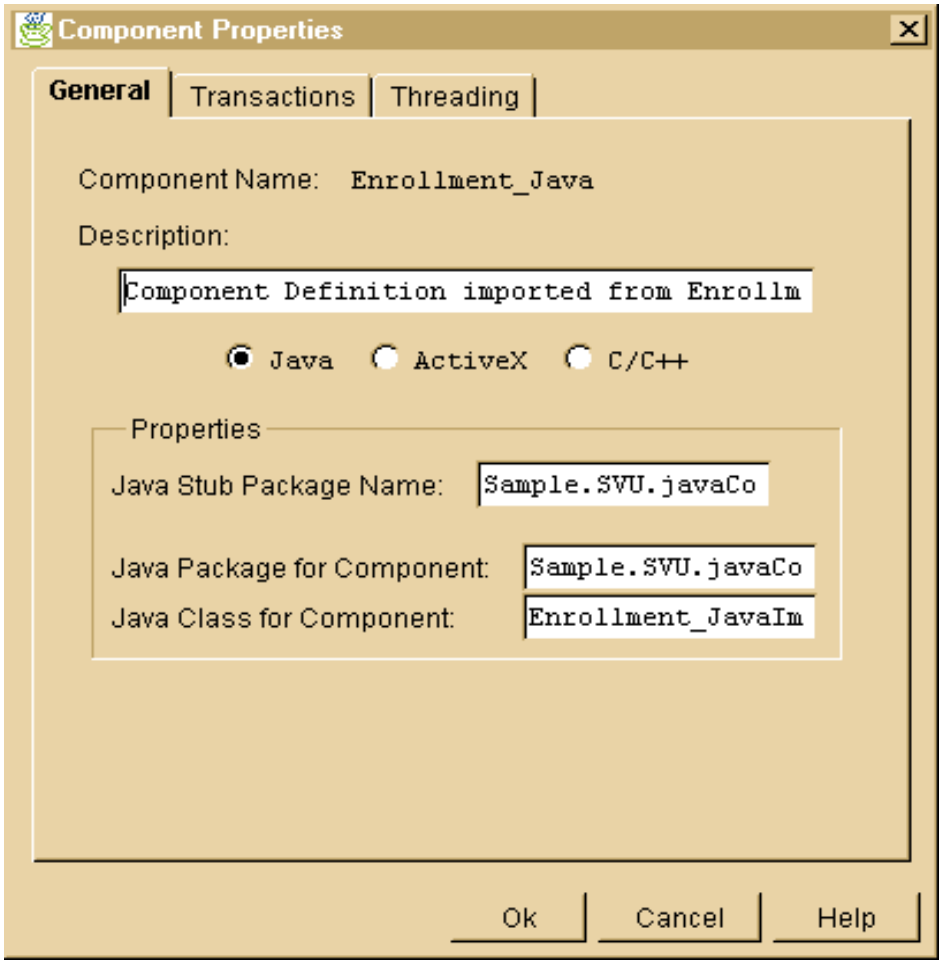


Figure 2: Registering a Component - Language choices

Two Tier vs N-Tier

Client/server applications, and even Java applications that call a database directly, represent the original, two-tiered application architecture. This architecture fits many needs, but often there is a penalty – the need to redevelop or copy code from one application to another as it is developed. More importantly, although Java is certainly a significant language for development, it's a recently developed one, and much of the logic that we need to use is written in other languages. Distributed applications, be they Web applets or standalone applications, also have a need to centralize logic, both to minimize the maintenance of the client, and to keep download time and response times reasonable.

Most of these issues have existed for years now, and the concept of externalizing and centralizing business logic from application logic developed into what is usually called a three-tiered or n-tiered architecture. CICS was arguably one of the first middle-tier solutions developed. Tuxedo from BEA has been available for many years as well. The concept is simple – remove common business logic from application code and place it in a central repository, where any application might call it. And one of the first things this middle-tier became charged with was handling transaction management. A transaction is a set of one or more database statements that must be treated as a unit of work. Simpler applications have no real need of transaction management, but applications that access multiple databases on different machines, perhaps in different locations, have a strong need for transaction management. This provides answers to the problem of two-phased commit, and of grouping transaction logic.

Middle Tier Servers

Over the years, a number of solutions have been tried, some more successful than others. In a mainframe world, CICS provides a certain amount of this functionality. In the client/server world, Tuxedo and other TP Monitors allow this type of development to occur. Unfortunately, the models that these systems present differ greatly from how we would like to view the world as Java programmers. Ideally, we'd like to be able to use the same syntax and constructs that we use for our application programming to talk to the Middle Tier. We'd like to view the logic in that tier as objects with methods, or sometimes as a single logical database, regardless of the number of real physical databases involved in the process. Previous solutions fell very short of this desire, which leads us to Jaguar CTS.

Note: The Common Object Request Broker Architecture (CORBA) specification provides another solution to this problem. Jaguar will eventually be CORBA-compliant so you will be able to leverage any Jaguar

work should you need to implement a CORBA solution as well.

Jaguar CTS

Jaguar CTS is a Component Transaction Server from Sybase, Inc. It's currently at version 1.1, which means it's still very new, and still has its share of undesired features. That's bugs to you and me. But in truth, Jaguar is also ten years old and represents a proven history of providing open solutions. So how can it be both? Jaguar is based upon Sybase's Open Client/Open Server architecture, mainly Open Server. Open Server was a set of APIs that allowed you to build business logic that could be called from the database and referred to in applications. Unfortunately, every time you made a change to the logic, you had to rebuild the Server and it didn't readily support a variety of programming languages or components.

Jaguar is the result of a concerted effort to take the power of Open Server and make it usable for the component builder, regardless of the language used. Unlike Open Server, Jaguar is a tool that allows you to register components, which can be developed in a number of languages and object models. In particular, Jaguar supports Java, C/C++ and Active X controls (C/C++, Visual Basic, PowerBuilder, Delphi, etc.). Native support for PowerBuilder code is in development, as is the ability to use CORBA objects. We'll focus here on what we can do in Java.

Product Highlights

Before we dive in, it's probably good to know a little bit about what Jaguar can do. First of all, Jaguar allows you to develop components which will be placed inside the server. Once there, Jaguar can provide access to these components to any language or product that can use a .JAVA file, a .C(PP) file or any product that can call stored procedures in a database.

Jaguar also provides several features that make it more than just an object broker. One of these is transaction management. We spoke a little about transaction management above. Jaguar makes it easy to group components into a transaction and specify which components participate. Jaguar also provides database connection caching, allowing connections to be reused and reducing the time it takes to establish a connection to a database.

Language Independence

One of the biggest advantages to Jaguar from a component standpoint is the ability to develop components in a variety of languages. Equally important is the ability to use Jaguar as a server from a variety of languages. The first point allows many companies to leverage existing code, usually in C or C++, with minimal changes. There are certain restrictions on what can be passed into and out of Jaguar and how methods are declared, but these will be lifted or reduced in version 1.5. For now, functions that Jaguar will export must be declared to return void. There's also a restriction on what you can pass in as parameters. Simple data types such as integer and string are fine. More complex types including arrays, structures and objects cannot be passed in. Probably the most significant impact this has is on result sets.

Methods in Jaguar can return a result set, which is one or more rows of data, plus enough metadata to describe the columns. Java has a number of classes that can act as result set consumers, or you can manually digest the result set (the metadata is the first row of the data in this case). Unfortunately, this is a one way street. You can't make changes to the result set and return them in bulk, or even as the row changes, at least not in the same manner that you would if you had obtained the result set directly from a database. This means that you'll end up writing insert, update and delete functions for your server objects so that they can handle changes, and code in the client objects that will know when to call the correct method. You'll probably have to keep track of the state of each row with a flag in order to accomplish this.

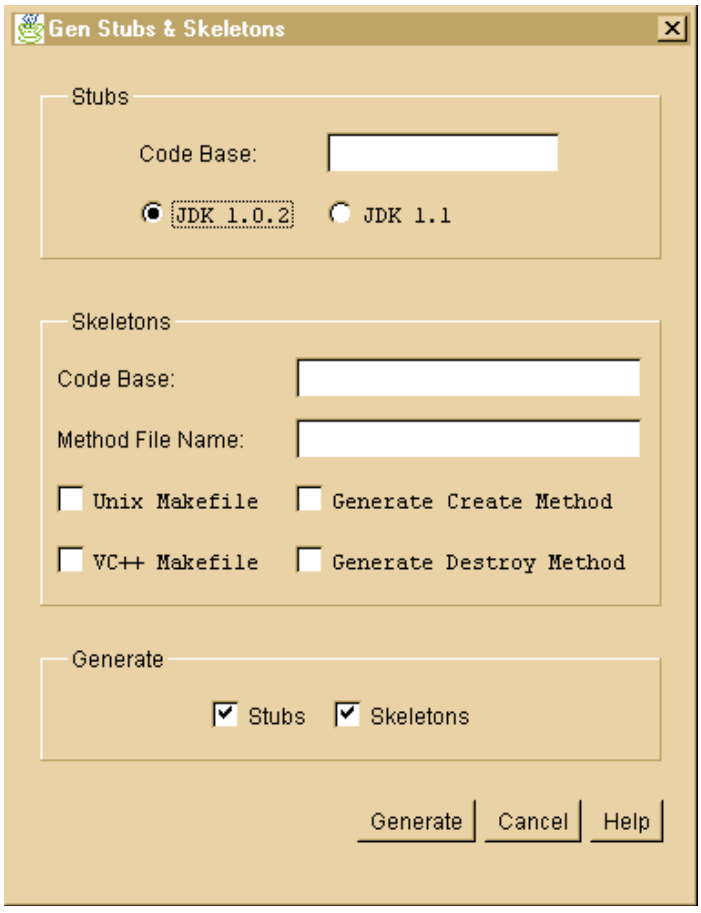


Figure 3: Generating Stubs/Skeletons

Transaction Monitoring

Transaction monitoring allows Jaguar to provide transaction coordination across multiple objects. Objects can be marked as needing to be part of a transaction, either a new one or one that is already occurring. Objects that are part of an already occurring transaction only create a new transaction if they are the first object called. Objects that require a new transaction always create a transaction for themselves.

Within a transaction, Jaguar keeps track of commits and rollbacks and intercepts these calls to the database. A real commit occurs only when the last object in a transaction executes a commit. If any object in the transaction performs a rollback, the entire set of statements for the transaction (including those made by other objects that issued a commit) will be rolled back. This allows you to code your components as if they were the only object in a transaction for testing, but use them with other objects in production.

Connection Caching

Jaguar also features connection caching. One of the biggest performance hits of a database application is the creation of a connection to the database. Since each object is distinct and will have its own connection, creating a series of objects could require a large number of connections, all using the same database parameters. Jaguar provides connection caching to minimize this work. If a cache exists that matches the database information of a requested connection, Jaguar does not perform an actual physical connection for each request. The first time through, a connection is established, but subsequent requests result in a logical mapping to the first physical connection. This results in faster setup of most components, but it does require setup for each user. Large numbers of users can make administration of the cache a difficult task, similar to the

Full pg

task of database administration. You should expect to have a Jaguar administrator who will work at least part time on creating connection caches and registering components.

Other Features

Jaguar has several other features that are more for convenience and organization than for any significant benefit to programming. The next four items are used to group functionality or determine availability of the various components to a particular user.

Servers

Jaguar can have several 'named' servers that run on a particular machine. By default, the first server is named Jaguar and has listeners on ports 8080 (http) and 7878 (queries and master). New servers can be created, but they must have different ports for the various listeners. Roles, packages and components can also be assigned to a particular server. This allows you to segment the functionality your users see at a very broad level. Assuming you had some overall concept of three categories of user (guest, ordinary, administrative), you could have each connect to a different named Jaguar server. Packages could be installed only as needed for each of these servers, effectively removing sensitive functionality from unauthorized users. (*Note*: Roles can be used to achieve similar functionality on a more granular level.)

Packages

A package in Jaguar is a set of components. By creating a package, you allow for a larger grouping of functionality, which makes it easier to manage components. By installing a package into a particular server, you allow access to all of the components in that package. It's similar to the concept of a ROLE in a database.

Components

Components are the objects that make up Jaguar. They have data and methods and can be grouped into packages. Components can be written in a variety of languages. Components cannot be placed into a server directly; they must be part of a package. The Jaguar manager provides a number of options for describing a component, including the methods of the component. It's also possible to declare a component and its method signatures prior to creation of the component. This allows the front end (client) programmers to begin coding prior to the completion of the server components.

Roles

Roles are optional in Jaguar, but they can be used in a similar fashion as ROLES in a database – namely, to restrict access to privileged functionality. Jaguar provides the ability to create users and assign them to roles, as well as the ability to assign roles to particular components.

Java and Jaguar

Now that we know a little bit about what Jaguar is, the question that should be on the tip of your tongue is why do I need it and how do I use it? I won't go into a long rationale for n-tiered architectures – either you need one or you don't. Given that you need an n-tiered approach, you might need Jaguar if you have a need for a variety of languages in the business logic layer, or you need the benefits of an object manager with transaction monitoring.

In the next two issues of ***JDJ***, we'll look at how to use this. We'll start by examining a simple but realistic server component next month. Then we will focus on how to use a Jaguar component (any component, it doesn't have to be Java, although in our example it will be) in a client application or applet.

About the Author

Sean Rhody is a respected industry consultant and a leading authority on PowerBuilder. He is also editor-in-chief of the ***PowerBuilder Developer's Journal*** *and one of the authors of "****PowerBuilder 5.0: Secrets of the PowerBuilder Masters.****" You can contact Sean at roadhog@nac.net*

roadhog@nac.net

1/2 Ad

Full pg

Becoming Friends with GridBagLayout

For the most precision in placing Components

by **John Tabbone**

My last column focused on several of Java's LayoutManagers, which are constructs used by developers to position Components within Containers using logic instead of pixel coordinates. We discussed all of Java's LayoutManagers except GridBagLayout, which is the focus of this article.

Of all of Java's LayoutManagers, GridBagLayout offers the developer the most precision in positioning Components. This ability comes at a great expense, as GridBagLayout is the most complicated (and probably the most poorly documented) LayoutManager in the AWT. GridBagLayout works integrally with a helper class called GridBagConstraints to place Components on the screen. Basically, a developer will set values in the GridBagConstraints object that specify things such as:

- Where the component will appear on the screen in relation to the other Components
- How tall and wide the component is
- How the component grows when the container resizes

A method provided in GridBagLayout will bind the constraints to a particular Component. When the Component is added to the Container (using the add method), GridBagLayout will use the information in the Components corresponding GridBagConstraints object to determine the Component's position and size.

In short, the process is:

1. Create Component
2. Set GridBagConstraints for Component
3. Bind Component and GridBagConstraints object
4. Add Component

Most of the work involved in using GridBagLayout is setting the constraints correctly.

As the name implies, GridBagLayout has something to do with a grid (and a bag). GridBagLayout is similar to GridLayout in the following regards:

- The Container is divided into a grid.
- Components are added to the cells of the grid.

The differences between GridLayout and GridBagLayout are more striking:

- Components do not necessarily fill their entire cell.
- All cells are not of equal size, meaning that all columns do not necessarily have the same number of cells, and all rows do not necessarily have the same number of cells.

Also, GridBagLayout offers no methods to explicitly define the number of rows or columns in the grid, nor does it have methods to define the size of each cell in the grid. Instead, GridBagLayout calculates the number of rows and columns in a grid by the number of Components placed on the screen. If a container has five Components lined up horizontally, then the grid consists of five columns and one row. If the Container has five Components lined up vertically, then the grid consists of one column and five rows. So how do you know how many columns and rows your GUI will have? Well, the best way is to hand draw your GUI and create the grid for yourself. Figure 1 shows a standard source/destination kind of GUI. Note the dashed lines and numbers. We have gone along the X axis and made a mark when we came into contact with the upper-left hand corner of a Component. We have done the same thing along the Y axis. Keep in mind when you do this on your own that you should mark only the upper left hand corner of the Component. After the exercise is complete, you are left with the grid your GridBagLayout will use to place Components.

Once the grid is complete, we have some very important pieces of information; the upper left hand coordinate of each Cell, and the width and height of each Cell. GridBagConstraints has data members that maintain this information and need to be set for the GUI to work. The data members are: gridx, gridy, gridwidth and gridheight. Look at Figure 2 to see the cells of the grid and Table 1 to see the GridBagConstraints attributes for each Component.

In our program, we have created a helper method, addComponent, to assist in setting the constraints. Notice that we do not have to create a new GridBagConstraints object every time we add a Component. Also notice the sequence of events: First we set constraints, then we bind the constraints to the Component using the setConstraints method and finally we add the Component. Be sure to do things in this

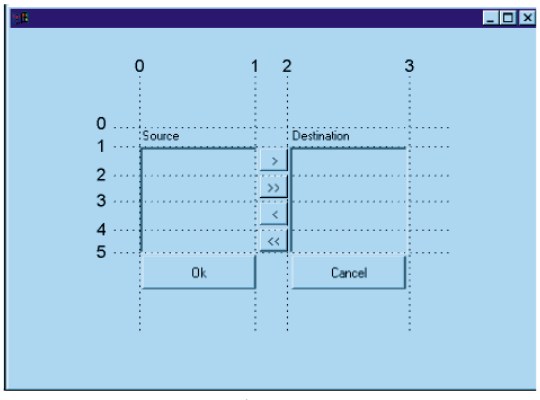


Figure 1

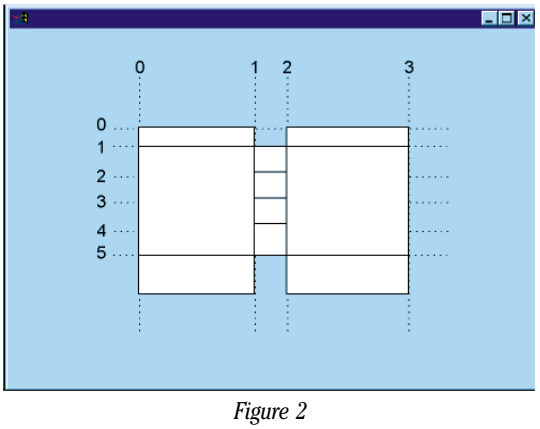


Figure 2

Full pg

order to avoid confusion! Look at Listing 1 for the complete code listing. And congratulations! If you can get this far, you have done the hard part.

Once the program has been compiled and executed we find that the results are not quite the same as our intention. The code needs some refinement. Look at Figure 3 for the output of the program. Notice that the four buttons in the center are not aligned, and everything is squished together in the center of the screen. The remaining tasks involve setting more constraints attributes on each Component.

GridBagConstraints.fill

The default value for the fill is NONE, meaning that a Component will not grow to fill its entire cell if there is extra space available in the cell. Look at the > button and the >> button. The width of the column is the width of the >> button (column width is always the width of the widest cell in the column). Using the fill attribute, we can specify that a Component will fill in one of four ways:

- GridBagConstraints.VERTICAL – The Component will grow taller, but not wider.
- GridBagConstraints.HORIZONTAL – The Component will grow wider, but not taller.
- GridBagConstraints.BOTH – The Component will grow both vertically and horizontally
- GridBagConstraints.NONE – The Component will not grow to fill extra cell space.

GridBagConstraints.anchor

The anchor attribute specifies where in the cell the component will be placed. The default value is CENTER. Other values are: NORTH, NORTHEAST, EAST, SOUTHEAST, SOUTH, SOUTHWEST, WEST and NORTHWEST. Look at Figure 4 and the code snippet in Listing 2 to see what happens to the labels and OK and Cancel buttons when the anchors are adjusted. Note that if the fill is set to BOTH, setting the anchor is pretty meaningless.

GridBagConstraints.weightx and GridBagConstraints.weighty

If you have taken the time to type in the code provided in Listing 2 and played around with the resulting Frame, you will have noticed that regardless of the size of the frame, the Components cluster in the middle of the Container and don't resize when the Frame resizes. Ascribing a weight to a Component will allow the Components cell to grow or shrink with the Container. Keep in mind that it is only the cell size that changes, not the Component in the cell. The default value for weightx and weighty is zero, meaning that extra space in the Container will not be absorbed by a cell. A non-zero weightx indicates the ratio describing how extra space will be distributed among horizontally neighboring cells. For example, if the OK button has a weightx of 2, and the Cancel button has a weightx of 1, when the Frame is made wider, the OK Button's cell will receive 2 pixels for every one received by the Cancel Button. The same is true for weighty neighbors. In general, the greater the weight, the more space the cell will receive.

Note also that weights affect the cell size, not the Component size. Here is where some of the other attributes of GridBagConstraints will play an important role. If the fill is set to something other than NONE, and a cell has a non-zero weight, the Component itself will grow to fill any newly acquired space in the cell. Alternatively, if an anchor is set to something other than CENTER, and a cell has a weight other than zero, the positioning of a Component within a cell will be more apparent. Play around with it!

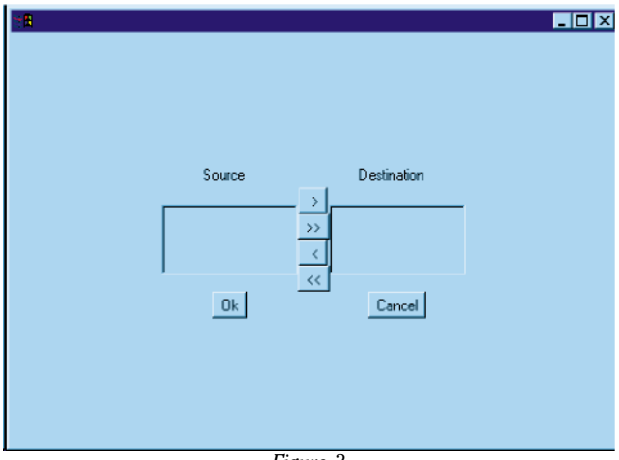


Figure 3

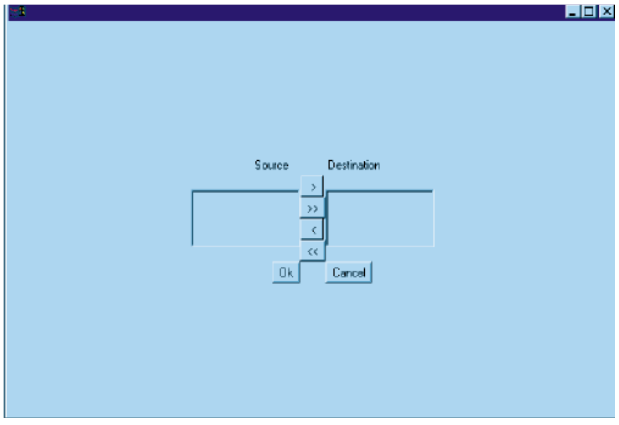


Figure 4

GridBagConstraints.insets

Insets add a border of a fixed pixel size to the inner perimeter of the cell. Adding insets may make the size of the Component shrink.

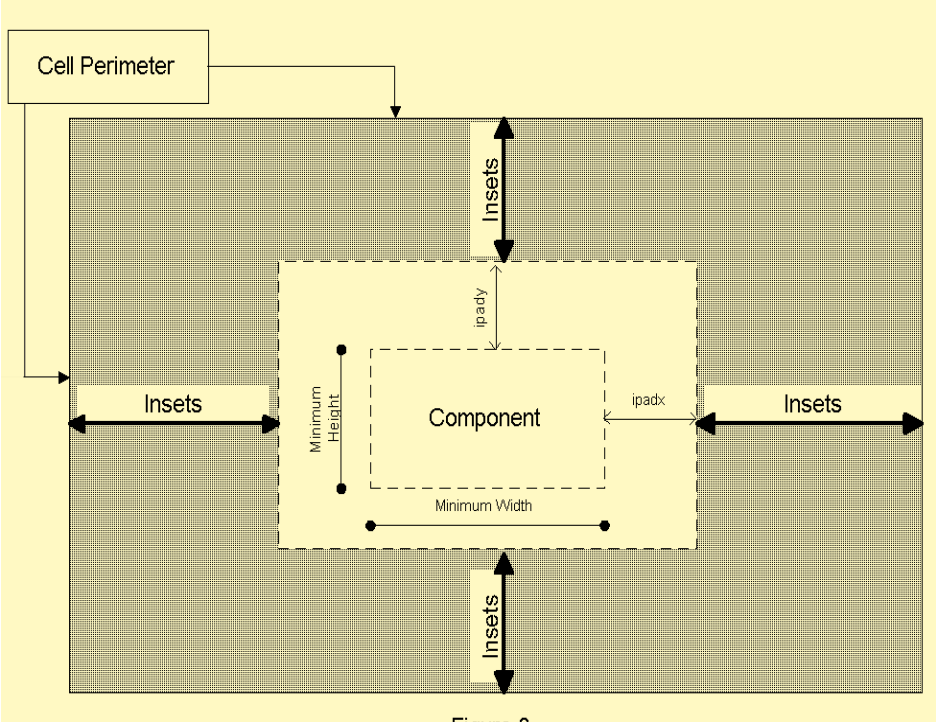


Figure 5

Name	gridx	gridy	gridwidth	gridheight
sourceList	0	1	1	4
destinationList	2	1	1	4
addButton	1	1	1	1
addAllButton	1	2	1	1
removeButton	1	3	1	1
removeAllButton	1	4	1	1
OKButton	0	5	1	1
cancelButton	2	5	1	1
sourceLabel	0	0	1	1
destinationLabel	2	0	1	1

Table 1

GridBagConstraints.ipadx and GridBagConstraints.ipady

ipadx and ipady (internal padding x and internal padding y) will add a fixed number of pixels to the width and height of a component. They will make the size of the Component larger. See Figure 5 to see the differences between insets and internal padding.

GridBagLayout Tips

Once you successfully develop a screen using GridBagLayout, you may want to change it later. This will be difficult if you have used consecutive numbers for your gridx and gridy values. For example, if you wanted to squeeze another Component in between the add Button and the addAll Button, you would have to enter new gridy values for all of the Components appearing below the new Component. Instead, it is a good practice to make your gridx and gridy values multiples of 10. Instead of column numbers being 1,2,3 they should be 10,20,30. The same is true for rows. A coworker has noticed that this is also a good practice in defining line numbers for BASIC programming.

I have not discussed GridBagConstraints.RELATIVE and GridBagConstraints.REMAINDER. Don't use them until you read my next column! As with any other topic in programming, the best way to learn it is to play around with the code, so please do so.

About the Author

John V. Tabbone is a lecturer at New York University's Information Technologies Institute, where he teaches two Java programming courses and advises on curriculum development. He has been a professional Java programmer since early 1996 and continues to consult on and develop systems for a variety of New York-based businesses. You may e-mail him with questions and comments at john.tabbone@nyu.edu

john.tabbone@nyu.edu

Listing 1.

```
import java.awt.*;

public class gbl1
{

 List   sourceList;
 List   destinationList;
 Button  addButton;
 Button  addAllButton;
 Button  removeButton;
 Button  removeAllButton;
 Button  OKButton;
 Button  cancelButton;
 Label   sourceLabel;
 Label   destinationLabel;
 Frame   aFrame;

 GridBagLayout  gbl;
 GridBagConstraints gbc;


 public gbl1()
 {
  sourceList  = new List();
  destinationList  = new List();
  addButton   = new Button( " >" );
  addAllButton  = new Button( ">>" );
  removeButton  = new Button( " <" );
  removeAllButton = new Button( "<<" );
  OKButton  = new Button( "Ok" );
  cancelButton  = new Button( "Cancel" );
  sourceLabel  = new Label( "Source" );
  destinationLabel = new Label( "Destination" );
  aFrame    = new Frame();
  gbl    = new GridBagLayout();
  gbc    = new GridBagConstraints();

  aFrame.setLayout( gbl );
  buildFrame();
  aFrame.setSize( 300,200 );
  aFrame.show();
 }


 public void buildFrame()
 {
  addComponent( 0,0,1,1,aFrame,sourceLabel );
  addComponent( 2,0,1,1,aFrame,destinationLabel );
  addComponent( 0,1,1,4,aFrame,sourceList );
  addComponent( 2,1,1,4,aFrame,destinationList );
  addComponent( 1,1,1,1,aFrame,addButton );
  addComponent( 1,2,1,1,aFrame,addAllButton );
  addComponent( 1,3,1,1,aFrame,removeButton );
  addComponent( 1,4,1,1,aFrame,removeAllButton );
  addComponent( 0,5,1,1,aFrame,OKButton );
  addComponent( 2,5,1,1,aFrame,cancelButton );
 }

 /**
```

```
 *A helper method to add Components to a Container using
* GridBagLayout
*/
public void addComponent( int x, int y, int w, int h, Container
aContainer, Component aComponent )
 {
  gbc.gridx = x;
  gbc.gridy = y;
  gbc.gridwidth = w;
  gbc.gridheight = h;
  gbl.setConstraints( aComponent, gbc );
  aContainer.add( aComponent );
 }



 public static void main( String[] args )
 {
  gbl1 myLayout = new gbl1();
 }


}// end class
```

Listing 2.

```
 public void buildFrame()
 {
  gbc.anchor = gbc.EAST;
  addComponent( 0,0,1,1,aFrame,sourceLabel );
  gbc.anchor = gbc.WEST;
  addComponent( 2,0,1,1,aFrame,destinationLabel );
  gbc.anchor = gbc.CENTER;           // back to default
  addComponent( 0,1,1,4,aFrame,sourceList );
  addComponent( 2,1,1,4,aFrame,destinationList );
  addComponent( 1,1,1,1,aFrame,addButton );
  addComponent( 1,2,1,1,aFrame,addAllButton );
  addComponent( 1,3,1,1,aFrame,removeButton );
  addComponent( 1,4,1,1,aFrame,removeAllButton );
  gbc.anchor = gbc.EAST;
  addComponent( 0,5,1,1,aFrame,OKButton );
  gbc.anchor = gbc.WEST;
  addComponent( 2,5,1,1,aFrame,cancelButton );
 }
```

Listing 3.

```
import java.awt.*;

public class gbl1
{

 List  sourceList;
 List  destinationList;
 Button  addButton;
 Button  addAllButton;
 Button  removeButton;
 Button  removeAllButton;
 Button  OKButton;
 Button  cancelButton;
 Label   sourceLabel;
 Label   destinationLabel;
 Frame   aFrame;

 GridBagLayout  gbl;
 GridBagConstraints gbc;


 public gbl1()
 {
  sourceList  = new List();
  destinationList  = new List();
  addButton   = new Button( " >" );
  addAllButton  = new Button( ">>" );
  removeButton  = new Button( " <" );
  removeAllButton = new Button( "<<" );
  OKButton  = new Button( "Ok" );
  cancelButton  = new Button( "Cancel" );
  sourceLabel  = new Label( "Source" );
  destinationLabel  = new Label( "Destination" );

  sourceLabel.setAlignment ( Label.CENTER );
  destinationLabel.setAlignment ( Label.CENTER );
  aFrame      = new Frame();
  gbl      = new GridBagLayout();
  gbc      = new GridBagConstraints();

  aFrame.setLayout( gbl );
  buildFrame();
  aFrame.setSize( 300,200 );
  aFrame.show();
 }

 public void buildFrame()
 {
 addComponent(0,0,1,1,gbc.HORIZONTAL,1,1,aFrame,sourceLabel);
 addComponent(2,0,1,1,gbc.HORIZONTAL,1,1,aFrame,destinationLabel
);
  addComponent( 0,1,1,4,gbc.BOTH,10,10,aFrame,sourceList );
  addComponent( 2,1,1,4,gbc.BOTH,10,10,aFrame,destinationList );
  addComponent( 1,1,1,1,gbc.NONE,1,1,aFrame,addButton );
  addComponent( 1,2,1,1,gbc.NONE,1,1,aFrame,addAllButton );
  addComponent( 1,3,1,1,gbc.NONE,1,1,aFrame,removeButton );
  addComponent( 1,4,1,1,gbc.NONE,1,1,aFrame,removeAllButton );
  addComponent( 0,5,1,1,gbc.HORIZONTAL,0,2,aFrame,OKButton );
  addComponent( 2,5,1,1,gbc.HORIZONTAL,0,2,aFrame,cancelButton );
 }

public void addComponent( int x, int y, int w, int h, int fill,
int xWeight, int yWeight, Container aContainer, Component aCompo-
nent )
 {
  gbc.gridx = x;
  gbc.gridy = y;
  gbc.gridwidth = w;
  gbc.gridheight = h;
  gbc.fill = fill;
  gbc.weightx = xWeight;
  gbc.weighty = yWeight;
  gbc.setConstraints( aComponent, gbc );
  aContainer.add( aComponent );
 }



 public static void main( String[] args )
 {
  gbl1 myLayout = new gbl1();
 }


}// end class
```

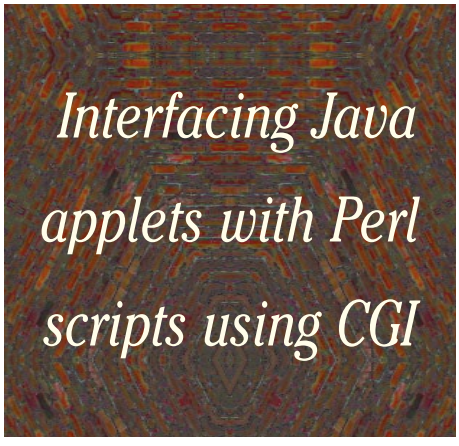
Don't Type it... Download it!

Access the source code for this and other articles appearing in this issue at JavaDevelopersJournal.com

BUILDING A Chat Applet

PART 2

by **Joseph DiBella**



Last month, we started a fun project in creating a chat room applet. The overall project illustrates how to create Perl scripts which will be used as back-ends for your Java applets. The Java applets will interact with the Perl program using a standard CGI interface. While you can have a lot of fun with this lightweight chat applet, my goal for you, however, is to look beyond this little applet and ask yourself, "How else can I use this method of sending data to a Web server, and process it on the Server side?". You can, for instance, use this to make a shopping cart, maintain data without using cookies, send a transaction to Cybercash (which uses Perl Scripts), etc.

This month, we will focus on the client-side Java applet. This is what the world will see as the chat room. To effectively do this, I have chosen to create an applet object and a Panel object. The applet will simply act as a container for the Panel object. The Panel object will then be the meat of the project.

The Chat class is derived from the applet Class. This class will be called by the HTML Web page, which can contain parameters that the applet will extract. The parameters will be used to set the path to the Perl program and name of the log file. The log file, which gets created and maintained by the Perl Script, will reside on the Web server. In it, we will store the last ten chat submissions in order of most recent first.

The init() method of the Chat class will use the ternary operator as a method to extract the parameters in the HTML file. Normally, I never use the ternary operator for anything since it is sort of cryptic. I use it here, however, because it lends itself to this purpose. The init() method also makes an instance of the ChatPanel object which will actually contain the chat engine. After setting the applet's layout to BorderLayout, I have added the ChatPanel object to it in its Center position (see Listing 1).

The start() and stop() methods just call the ChatPanel object's start() and stop() methods, respectively. This will be important not only for starting and stopping the chat thread, but for sending a message to the Web server that the client has entered/exited the chat room.

```
public void start(){
    chatScreen.start();
}
public void stop(){

    chatScreen.stop();
}
```

Finally, I have declared an accessor method to get the cgiPath, called getCGIPath(). This method will be called from within the SubmitToChatServer objects.

```
    public String getCGIPath(){
        return cgiPath;
    }

    ChatPanel chatScreen;

    Thread runner = null;
    String chatLogFileName, cgiPath;
}
```

The ChatPanel class is quite complex. It implements Runnable, since we will use a Thread to refresh the chat data on our screen. This object will extend Panel, which we will configure as a CardLayout container. It will contain two additional Panels, one of which will gather setup information, such as the client's desired chat name, and the other will display the actual chat messages.

To construct this object, we will pass two parameters:

- A handle to the applet
- A String that will specify the file name of the chat log

After initializing some instance variables, we construct the panels. To build them, I have opted to nest some additional components, such as Panels, instead of using GridBagLayout. Both the Setup and Chat panels use BorderLayout. This

method adds the various AWT components needed to these panels using the standard add() method. After each Panel is constructed, we will add them to the card layout panel, which will contain them (see Listing 2).

The start method checks to see if the user has entered a name. It the user has not entered a name to chat with, it will bring the setup panel to the front and request focus for the TextField, which will be where the user will enter a name. If the user has entered a name, we will create a new SubmitToChatServer object with a message indicating that this person has entered the chat room. On the same line, we start() that object, which will, in turn, spawn a new Thread to send that data to the chat server. Each line we send to the chat server will work exactly in this way, spawning a separate thread with which to send the data. Next, we will start our chat engine by creating another thread for it, if one does not already exist, and starting it. This will, in turn, invoke this object's run() method in the thread (see Listing 3).

The stop() method will do a similar thing to what the start() method does. It will create another SubmitToChatServer object with a message indicating that the user has left the chat room and start the process of sending it in another thread. After doing so, it will stop the chat engine's thread so that we will not be constantly refreshing the screen.

```
    public void stop(){
        if(runner != null){
            new SubmitToChatServ-
er("*** "+identity+" Has Left This
Chat Room. ***",app,
chatFileName).start();
            runner.stop();
            runner = null;
        }

    }
```

The run() method will be the heart of the chat engine. The first thing we do is request focus on the TextField, which will be used for chat entry. The next part of this method required some thought and experimentation. Since the AWT's TextArea object does not wrap text to the next line when we have reached the right side of it, I needed to develop my own text wrapping technique. In order to do this, we need to figure out the width, in pixels, of the average character in a specific font. I did this by using a FontMetrics object for the chatArea. I determined the width of a string which contains the alphabet in upper and lower case. I have even included some spaces and periods. Then, I divided the length of this string by the number of characters in it. This gives me the average width, in pixels, of the text that users may type in. I can later compare this to the width of the Text area to determine how many characters will fit on a line. Once this is done, we go into an infinite while loop, which invokes a method called refresh() and pauses the thread for the length of time specified in refresh time (see Listing 4).

As you may have guessed, the refresh() method will be used to refresh the chat data on the screen. It will be responsible for downloading the chat log file and parsing its information. First, it creates a URL object for the chat log file. Next, we create a DataInputStream object, which we will use to read its data. Since the log file stores its most recent data first, we will need to read each line and pre-pend it to the front of a temporary String, named chatData, which will be used to store the new data with the oldest data first. The first line we read (Most Recent Chat) will be stored for the next time this method is called. Each line read will need to be processed by the processText() method. This method will perform the wrapping of text if necessary. We will keep doing this on a line by line basis, until we read a line that matched the first line read from the previous iteration of this method. When finished, we will append the temporary String to the chatArea TextArea and close the stream. To force the TextArea to scroll down to the bottom, we will select the last character of text contained within it (see Listing 5).

"How else can I use this method of sending data to a Web server, and process it on the Server side?"

The processText() method is responsible for word wrapping within the TextArea. It takes the unprocessed text in as a parameter and returns the processed text as a String. If the text contains fewer characters than the average characters per line, then we will simply return the text with two new-line characters appended to it. If the text contains more characters than the average characters per line, then we examine the first group of characters that fit on a line. We count backwards until we find the last instance of a space. Here, we will insert a new-line character and repeat the process on the remaining text after we pre-pend the person's name, which we extract from the text, to the beginning of the remaining text. When finished, we return the completed processed text with two new-line characters appended to the end (see Listing 6).

Since most browsers in use today do not support 1.1, I still use the 1.0 event model for applets. The handleEvent method is used for both the setup and chat panels. Here, we simply handle the action events for the buttons, as well as if the person generates an action event on a TextField (by pressing the enter key, for instance). If the target is the chatLine or the sendButton, we call the sendChat() method. If the target is the nameField or the setupButton, we need to check if they have actually entered a name. If not, we tell them to enter a name. If they have entered a name, we enter the chat room, show the chat panel and start the chat engine (see Listing 7).

The sendChat() method simply creates a new SubmitToChatServer object with the text to be sent to the Perl Script. The text is a String constructed with the identity of the person and the text extracted from the chatLine TextField. We need to start the transfer, so we invoke the start() method on this object, which will send the chat line in a new Thread. Next, we need to clear the chatLine TextField.

```
    void sendChat(){

        new SubmitToChatServer(iden-
tity+" >: "+chatLine.getText(),app,chatFile-
Name).start();
        chatLine.setText("");

  }
```

To finish off this class, I have written a number of accessor and mutator methods. These include:

```
setRefreshTime()
setIdentity()
getIdentity()
setChatLogName()
```

I may use these methods later if I wish to be more elaborate in my applet class (see Listing 8).

There may be a number of things you wish to do to optimize this Chat applet. For instance, you may want to move the refresh time up or down depending on your chat traffic. Remember, however, that you will be

refreshing at longer or shorter intervals and each refresh will read the current log file, which stores up to twenty lines of chat. If you have heavy traffic in your chat room, more than twenty lines may have been submitted during that time period. You may want to alter the Perl Script to store more lines; however, twenty seem to be more than adequate in most situations.

In addition, I have noticed that performance is drastically improved when using ISAPI Perl with IIS on Windows NT Server. You can download this for free at http://www.activestate.com. *Note:* You will need both the Win32 and ISAPI versions. The ISAPI version must be installed on top of the Win32 version. You may need to edit the registry, particularly in the scriptmap section under HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\W3SVC\Parameters\Scriptmap. Here, you would want to make sure the Perl is registered to the file extensions .pl and .cgi. The .pl should be set to C:\Perl\bin\perlIS.dll and the .cgi should be set to C:\Perl\bin\Perl.exe %s %s. If you want to use the ISAPI version, rename your Perl Script's extension to .pl and make this modification in the Java program as well, where the Perl Script is referenced.

Well, that's it. I hope you have had a lot of fun with this project. It demonstrates several exciting methods of interacting with a Web server and introduces many new possibilities for your applets.

About the Author

Joseph M. DiBella is the Senior Java Instructor and Curriculum Developer for Computer Educational Services in New York City. He also is the President of HMJ Electronics, a computer consulting company which develops software and Java-enhanced Web sites. Joe can be reached at lite-n-sweet@java-joe.com

lite-n-sweet@java-joe.com

Listing 1.

```
    public void init() {

        super.init();

chatLogFileName = getParameter("logfile") != null ? getParame-
ter("logfile") :"chatlog";

        cgiPath = getParameter("cgipath") != null ? getParame-
ter("cgipath") : "cgi-bin/";

chatScreen = new ChatPanel(this,chatLogFileName);

        setLayout(new BorderLayout());
        add("Center",chatScreen);

    }
```

Listing 2.

```
public class ChatPanel extends Panel implements Runnable{

    ChatPanel(Applet app,String fileName) {

        super();
        this.app = app;
        chatFileName = fileName;
        //Build the Setup Panel
        Panel setup = new Panel();
        Panel setupNorth = new Panel();
        Panel setupCenter = new Panel();
        Panel setupSouth = new Panel();
        setup.setBackground(Color.cyan);
        setupButton = new Button("Enter Chat Room");
        nameField = new TextField(20);
        nameField.setBackground(Color.white);
        setupCenter.add(new Label("Enter Name:"));
        setupCenter.add(nameField);
        setupCenter.add(setupButton);
        setupNorth.setFont(new Font("TimesRoman",Font.BOLD,36));
        setupNorth.add(new Label("Welcome to Chat"));
        setupSouth.setFont(new Font("TimesRoman",Font.BOLD,36));
        setupSouth.add(new Label("You Must Enter A Chat Name"));
        setup.setLayout(new BorderLayout());
        setup.add("North", setupNorth);
        setup.add("Center", setupCenter);
        setup.add("South", setupSouth);
        //Build the Chat Panel
        Panel chat = new Panel();
        chat.setBackground(Color.black);
        chat.setLayout(new BorderLayout(10,10));
        northPanel = new Panel();
        southPanel = new Panel();
        northPanel.setBackground(Color.cyan);
        northPanel.setFont(new Font("TimesRoman", Font.BOLD,24));
        northPanel.add(new Label("Joseph DiBella's Cyber-Chatter-
box"));
        chat.add("North", northPanel);
        chatArea = new TextArea("Welcome to Joe DiBella\'s Chat
Applet\n\n");
        chatArea.setEditable(false);
        chatArea.setBackground(Color.cyan);
        chatArea.setForeground(Color.black);
        chatArea.setFont(new Font("Helvetica", Font.BOLD,14));
        chat.add("Center",chatArea);
        chatLine = new TextField();
        chatLine.setBackground(Color.white);
        sendButton = new Button("Send");
        southPanel.setLayout(new BorderLayout(5,5));
        southPanel.add("Center",chatLine);
        southPanel.add("East", sendButton);
        chat.add("South", southPanel);

        clayout=new CardLayout();
        setLayout(clayout);
        add("Setup", setup);
        add("Chat", chat);


    }
```

Listing 3.

```
    public void start(){
        if(identity != null){
            new SubmitToChatServer("*** "+identity+" Has Entered
This Chat Room. ***",app,chatFileName).start();
            if(runner == null){
                runner=new Thread(this);
                runner.start();
            }
        }else{
            clayout.show(this,"Setup");
            nameField.requestFocus();
        }
    }

Listing 4.
    public void run(){

        chatLine.requestFocus();
        // Figure out line length in pixels
        FontMetrics fm = chatArea.getFontMetrics(chatArea.get-
Font());
        aveChar = fm.stringWidth("ABCDEFGHIJKLMNOPQRSTUVWXYZabcde-
fghijklmnopqrstuvwxyz  .. .. ")/62;
        aveCharsPerLine = this.size().width/aveChar;

        // Start the chat engine
        while(true){
            refresh();
            try{
```

AD

```
                Thread.sleep(refreshTime);
            }catch(InterruptedException e){}

        }

    }
```

Listing 5.

```
void refresh(){

            try{
                log = new URL(app.getDocumentBase(),chatFile-
Name+".log");

                DataInputStream din = new
DataInputStream(log.openStream());

                String tempLastLine=lastLine;
                String in="";
                String chatData = "";
                boolean first = true;
                boolean finished = false;

                while((in=din.readLine())!=null){

                    if(in.startsWith(lastLine)){
                        finished = true;
                    }
                    if(!finished){
                        if(first){
                                tempLastLine=in;
                                first=false;
                        }

                         chatData = processText(in)+chatData;
                    }

                 }

                  chatArea.appendText(chatData);
                  lastLine=tempLastLine;
                  din.close();

            }catch(Exception e){
                System.out.println("Error in refresh:" +
e.toString());

            }
            // Force the Text Area to scroll to the end

chatArea.select(chatArea.getText().length(),chatArea.getText().leng
th());


    }
```

Listing 6.

```
    String processText(String text){
        /* This method is responsible for word wrapping in the
text area. If a line is longer than
           the width of the Text area, we will insert a new-line
chatacter in place of a space */
        int textLength = text.length();
        if(textLength <= aveCharsPerLine){
            return text+"\n\n";
        }
        // We will need the name of the person if we need to
wrap. This extracts that info
        String chatName = text.substring(0,text.indexOf(">:")+2);
        String processedText = "";

        while(text.length() > aveCharsPerLine){
            int i = text.lastIndexOf((int)' ',aveCharsPerLine);
            try{
                processedText += text.substring(0,i)+"\n";
                text = chatName + text.substring(i);
            }catch(Exception e){/* code for exception*/
                System.out.println("Error processing text:" +
e.toString());

            }
        }

        return processedText+text+"\n\n";

    }
```

Listing 7.

```
    public boolean handleEvent(Event event) {
        if (event.id == Event.ACTION_EVENT && (event.target ==
sendButton||event.target == chatLine)) {
                sendChat();
                chatLine.requestFocus();
                return true;
        }
        if (event.id == Event.ACTION_EVENT && (event.target ==
setupButton||event.target == nameField)){
             if(nameField.getText().equals("")){
                app.showStatus("You must enter a name");
                nameField.requestFocus();

            }else{
                app.showStatus("Entering Chat Room");
                setIdentity(nameField.getText());
                clayout.show(this,"Chat");
                start();
            }
            return true;
        }

        return super.handleEvent(event);
    }
```

Listing 8.

```
    public void setRefreshTime(int i){
        refreshTime = i;
    }

    public void setIdentity(String i){
        identity = i;

    }
    public String getIdentity(){
        return identity;
    }
    public void setChatLogName(String i){
        chatFileName = i;
    }

    //{{DECLARE_CONTROLS
    private TextArea chatArea;
    private TextField chatLine;
    private Panel northPanel, southPanel;
    private Button sendButton;
    private Applet app;
    private int refreshTime = 10000;
    private int aveChar, aveCharsPerLine, chatSize;
    private String identity = null;
    private String chatFileName = "chatlog";
    private URL log;
    private String lastLine=" ";
    private Thread runner;
    private Button setupButton;
    private TextField nameField;
    private CardLayout clayout;
        //}}
}
```

CocoBase Enterprise

by THOUGHT, Inc.

Turn Your Java into a Database Access Powerhouse!

by **Ed Zebrowski**

Trying to develop applications in conjunction with a database can be a nightmare. Customer orders must be filled, accounts payable must be debited and inventory must be adjusted. Performing a task like this directly from the Internet lends itself to the flexibility and platform independence of Java.

Wouldn't it be nice if it was as simple to use elements of a database as objects in your Java applications, threading them directly to the Database? Well the application you've been dreaming of is here.

CocoBase Enterprise from THOUGHT, Inc. is a powerful new point-and-click Database Access language creation and management tool. It is Pure Java, a snap to use and best of all, it's 100% scalable.

System Requirements

CocoBase Enterprise runs on any JDK 1.1x or JDK 1.02 enabled system. I ran it over JDK 1.1.5 on a 150MHZ Pentium with 16MEG and WIN95, and it performed fine.

Installation

Installation of my demo version was initiated at the command prompt. Making the installed directory current, I needed to type only: java install_cocodemo3tier11_1027_timeout1201. A GUI interfaced wizard then opened, making installation effortless.

THOUGHT Inc. provides a test program called COCODEMO to verify if the installation was done properly. This demo applet needs the application server to attach itself to. Thus, it's necessary to start the three-tier CocoBase Enterprise Application Server by adding the CocoBase home directory\demos to the CLASSPATH, then typing the path to the startserver.bat file. Another command window opens, typing the command to start the cocodemo.bat file. This opens a customer demo applet. By typing "My Customer" in the input window and then clicking "Find", an existing sample of customer data should appear in the applet window. This automatically performs and logs the request, which will appear in the server window, not the client.

A Trial Run

Once you have CocoBase Enterprise up and running, the best way to showcase the application's many attributes is to go through a trial run. Open a command prompt and execute the proper demo batch file. The file you execute varies depending upon which JDK you're running. If done successfully, this will open a login applet screen with user name and password already filled in. Merely clicking the "login" button gets you started.

Let's try running through the "SELECT" operation on object schema: "Customer". Click File-Open to view the object schemas that come with this demo. Click "Current Data for Select Fields" grid. Select the "Name" field by clicking on that row. Click "View"-"Select"-"Clauses". This will display the "Current Data For Select Conditions" grid. Select the first data row of the grid by clicking on that row. Click "Grid" - "Update" to view how the condition is defined. Click CANCEL after viewing. Click "View" -"Show SQL" -"Show Select" to view the constructed "SELECT" SQL statement for this object schema. This very same process can be run through for the "Insert" and "Update" operations as well.

Application developers no longer need to concern themselves with learning the database access language, but can spend their time on developing specific applications. CocoBase can handle full table spanning select, updates, inserts, deletes and procedure calls without recompiling of the client-side Java application.

CocoBase Enterprise
THOUGHT, Inc.
657 Mission Street, suite 202
San Francisco, CA 94105
Phone: 415 836-9199
Fax: 415 836-9191
Web: http://www.thoughtinc.com
Email: cocobase.support@thoughtinc.com

Other Features of CocoBase Enterprise,

- Object binding for Mainframe, Object and Relational databases: This means that unlike some other products in this space, customers are not restricted to using a Relational Database as their only source of data
- Speed: Performance has been clocked up to 30 times faster than standalone type 3 or type 4 JDBC drivers. This is possible because the CocoBase API sends only the application objects and no meta data between client and server.
- Enterprise scalability: CORBA or RMI is used as a transport layer for the objects. Adapters have been developed for the different transport layers that act as plug-ins to the system at run time.
- Distributed Architecture
- Connectivity to standard Java Development Environments with JavaBeans™ (Bean source code included)
- Upgrade from 2-tier to 3-tier implementation without recompiling
- Ability to use the same Java application across multiple databases without recompiling
- Ability to generate relational tables from existing Java code
- Ability to generate commented Java code from existing relational tables including foreign key relationship code
- Ability to integrate existing Java, C and C++ code into the Application Server

If you've been scorched by the task of developing Java applications from a database, try using CocoBase Enterprise to cool and soothe the burn!

JavaHelp™ Software

New API for Developing Online Help from Sun Microsystems

by **Nancy Lee**

Most interactive applications require online help – Java applications are no exception. To address this need, Sun Microsystems, Inc. has developed the JavaHelp API and help system. The JavaHelp software fills a void in the help market for a cross-platform, browser-independent help format that will eliminate the need to develop costly proprietary Java technology-based help systems or settle for a platform-dependent solution that will require you to rewrite your help system for each platform.

Written entirely in the Java programming language, the JavaHelp help system provides capabilities for navigating, searching and displaying help information, making it easy for an end-user to acquire knowledge about how to use an application or applet. Using the JavaHelp software, developers and authors are able to incorporate online help for components, applets, applications, operating systems, devices and Web pages. Developers can also use the JavaHelp software to distribute online documentation in a corporate intranet or Internet.

The JavaHelp system offers a full-featured help system, including its own help viewer, which consists of a toolbar, content pane and navigation pane. The content pane uses HTML 3.2 as its format for displaying topics and can run Java applets. The navigation pane provides the table-of-contents (TOC), index and full-text search navigational controls. The TOC supports a collapsible and expandable display of topics and an unlimited number of hierarchical levels. The file formats for the navigational controls are all XML-based.

Through the JDK internationalization feature, the JavaHelp software will support internationalization. Localization of help content, indexes and table of contents are supported and are loaded in a locale-specific way.

The JavaHelp system, designed to be very flexible, can be displayed in its own primary window or embedded in an application. In fact, because the JavaHelp system is based on the Java Foundation Classes (JFC) components, the navigational controls and the viewer can be individually embedded in an application. In addition, help projects can be merged using multiple TOCs and indexes.

Packaging help information is also very flexible with the JavaHelp software. The JavaHelp system information is delivered in a JAR file, which compresses and encapsulates the help files into a single file. The JavaHelp system can extract help information files from the JAR file as they are required.

The flexibility and ease of integration of the JavaHelp API makes it easy for developers to customize and extend the user interface and functionality. It allows custom or third-party navigational controls, search engines and content viewers to be added. In addition, applets can be used to extend the help system with additional functionality. The JavaHelp system will provide some applets, including applets for pop-up and see-also windows.

Designed for the Java platform, the JavaHelp software will work especially well with networked applications. The help data and search functionality can reside on either the client-side or server-side. Help data and new functionality can be dynamically updated over the web. Coupled with its cross-platform benefits and network design, the JavaHelp software is ideal for use in a heterogeneous environment, such as the Internet or corporate intranet.

There are other help formats available today, both Java technology-based and non-Java technology-based formats. However, none meets the needs of an industrial strength, cross-platform, browser-independent solution that Sun is offering through the JavaHelp API. Many of the other existing Java technology-based help formats are proprietary help systems or developments done by smaller companies that may not offer extensive support for their help systems. In addition, many of these formats are inadequate for supporting a sophisticated, complex help system that is needed by large, complex applications. In regards to help formats that are not written in the Java programming language, such as WinHelp and HTML Help, they do not offer the benefits of the JavaHelp software, as they are dependent on a specific platform and browser. Incidentally, many of the proprietary Java-based help systems are moving to support JavaHelp API. For example, Sun's Java Workshop™ help, a customized help system specific to Java Workshop product, plans to migrate to the JavaHelp system in a future release.

One of the biggest requests from customers for developing a JavaHelp system is having help authoring tools available. Because the JavaHelp software is being developed by an open, industry-participative process, it has received widespread support from the industry and help authoring tool vendors. The leading help authoring vendors have publicly announced support for JavaHelp, including Blue Sky Software, ForeFront, Wextech Systems, Creativesoft, Quadralay, Virtual Media and HyperAct. These authoring tools will make it easy for authors to develop the JavaHelp system by offering a simple conversion utility to generate the JavaHelp help format.

Sun has announced that the JavaHelp software will be delivered as a standard extension to the next release of the JDK. An early access release of JavaHelp software will be available by early April. At that time, developers can evaluate and use the JavaHelp software, with the caveat that authoring support will not be available at that time. The JavaHelp software is scheduled to ship this summer. Redistribution of the JavaHelp binaries will be permitted, royalty free with your product.

For more information about the JavaHelp software, visit the Web site at http://java.sun.com/products/javahelp.

About the Author

Nancy K. Lee is Product Manager for JavaHelp within JavaSoft, a division of Sun Microsystems, Inc. Since joining Sun in 1995, she has also worked in the JavaBeans™ product marketing group and helped found the Java Developer Connection program.

InstallAnywhere
by Zero G Software

Build custom installers for all of your Java applications

by **Ed Zebrowski**

The huge task of creating that great Java application is finally over. Although the product is "user friendly" when up and running, installation has typically been very cumbersome and time consuming. A product that is easy for the layman to install will surely outsell one that isn't.

You must now go back to work. What's needed is a development tool that not only enables applications to be installed easily for the end-user, but is quick and easy for the developer to use as well. What's needed is InstallAnywhere from Zero G Software, Inc. InstallAnywhere enables the creation of a single, universal installer that will operate on any Java-enabled platform. This includes Windows, UNIX and Mac OS. IA tailors itself to the user's system, saving the time and expense of building separate installers for each platform.

Installing IA was an absolute breeze. Anyone who can install basic software off a CD can have IA up and running in a matter of minutes. If you're like me, typing code at a command line all day will start to drive you batty. Here lies another wonderful feature: IA runs entirely in a GUI interface.

When I first opened IA, I was given two choices; an "open existing" option which allowed me to continue working on a previous installer, or a "new" option, which asked me to name the new installer. Upon clicking new, a "quick start" feature greeted me. It then walked me through a seven-step shortcut which made building my installer remarkably easy. The seven steps I had to follow were:

1. Select the "Add files/folders" button
2. Select and add the files to install
3. Select the main class
4. Create a launch Anywhere executable (for Java applications)
5. Switch to the build task (for Java applications)
6. Press the "build installer" button
7. Run my installer

Figure 1: InstallAnywhere's Project Wizard walks the user through the basics of building an installer, even automating tasks like setting your application's classpath.

InstallAnywhere
Zero G Software
118 King St. #415
San Francisco, CA 94107
Phone: 415-512-7771
Fax: 415-512-7775
Web: www.ZeroG.com
Email: info@ZeroG.com
Requirements: IA runs on any Java 1.1-enabled platform
Price: Express Edition: $495, Standard Edition:$995, Commercial Edition: $3995

If a simple installer is required, I've found it! I came away amazed at just how easy it is to use this product. If your application requires more complex interactions, fear not. Although the seven-step method can't be used for more complex installations, life will still be much easier using IA.

When I bypassed the "quick start" feature, I thought I was looking at a nicely laid out Web page rather than a powerful application builder. On the left side of the screen, there are seven tasks from top to bottom. When a task is clicked on, a center work field is filled with the various tools pertaining to that task. Let's take a look at each task, and what's involved with each one.

Files

This task displays all the files and directories that will be installed. It is broken down into three panes:

- File "Hierarchy" and assignment to "Bundles" on the top of the screen. This displays all files and folders that are part of the project. The icons on each file or folder indicate what item will be installed or what action will be taken during the installation process.
- At the bottom left of the pane is the "Install File/Folder Action Customizer". This allows adjustment of how the file, folder or archive will be installed.
- Rules List and Customizer: This allows the developer to assign or design cus-

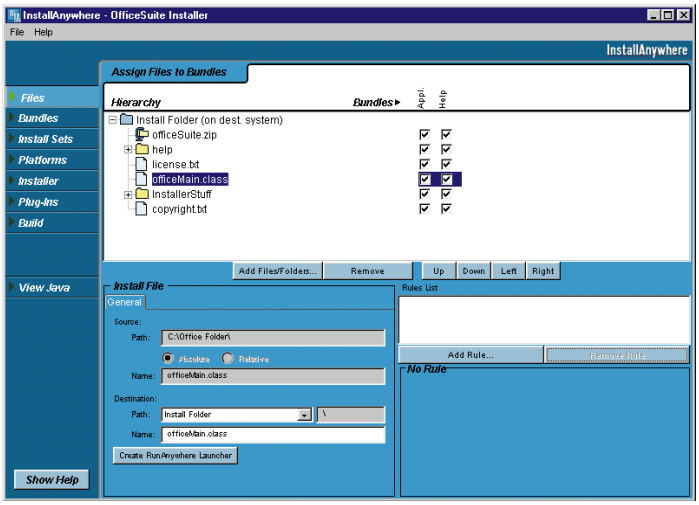


Figure 2: IA looks more like an Web page than a powerful builder!

Figure 3

tom "rules" or a set of conditions that must be met for an item to be installed.

Bundles

This task shows the "Bundles" (functional groups of files) in the installer, along with the "Install sets" each bundle is assigned to. This task also has three panes:

- The Bundles "Hierarchy" and assignment to "Install Sets." These are used mostly for custom installations so the end-user can choose which groups of files to install.
- The Bundle Customizer: Shown in the lower left pane, this allows adjustment of various bundle settings.
- Rules List and Customizer: Allows the same assignment of rules to bundles as to files, as described in the previous section.

Install Sets

This task shows the groupings of bundles that comprise the entire installation in your installer. It also has three panes:

- Install sets list: Shows every install set in your installer. Allows settings for typical and minimal installations, as well as full adjustments to set the defaults for an installation.
- Install Set Customizer: This setting allows the customizing of four install set settings. Name and Short Name assign a name and abbreviated name to identify your install set. Description provides descriptive help about an install set.
- The Button section identifies the icon displayed inside each install set. A GIF or JPEG that is 32x32 pixels can be used.
- Rules List and Rules Customizer: Allows you to assign criteria (rules) to any install set.

Installer

This task allows you to specify five options about the installer being designed:

- The Operation menu allows you to modify certain basic settings in each installer.
- The Rules menu assigns criteria that must be met for the installation to run.
- The Appearance menu allows the assignment of a name and appearance of your installer.
- The Steps menu assigns the steps to include in the end-users installation process. One option here allows you to choose to install a Java Virtual Machine specifically for the application being installed or to use an existing JVM.
- The Billboards menu allows you to specify which billboards will be shown and what order they will appear in. Billboards will appear during end-use installation. They may be used to highlight features of the application being installed, or they may provide a set of tips and reminders to the end user. I found this to be a handy feature.

Plug-ins

This task brings up the interface for the IA plug-ins you may be using. Zero G tells us that there are a number of plug-ins under development, including a multi-language kit.

Build

The Build task allows the specification of what types of installers you want to build and which platforms you want your installer to run on. One of the specifications of this task is the "Delivery" option, which specifies which type of installation you want to build. IA supports two types of installations:

- CDROM/File server: This creates a folder for each specified platform. Inside each folder is a double-clickable "LaunchAnywhere," an executable that will run the Java installer. Use this type of delivery to build an installer that will be executed from a CD-ROM or a floppy disk. The folder structure is written out to any ISO 9660-compatable CD-ROM drive.
- Web downloads: This creates a clickable self-extractor for each platform. These contain a copy of the installation with the correct VM for the platform. They can be placed on a server for download.

View Java

This task will generate Java code for the installer you have just built. By looking at this, you can see how objects are created and get a good picture of how your installer will work. This is a good feature for the true Java-heads who are just not comfortable unless they can look directly at their source code. The code faithful will be disappointed to learn that the code must be edited outside of IA. Files to be edited must be copied and then opened in an external development environment. The InstallAnywhere Standard Edition, it should be noted, does not support the addition of code outside of the existing IA objects. You would have to move up to InstallAnywhere Commercial Edition, an additional $3000.00 cost, to have this feature. For the extra three grand, you get detailed developer and API documentation and sample code and can use JavaBeans.

Conclusion

The job of the software developer is to develop products that are easy for the customer to install and use. In today's "we need it yesterday" world, it's good to know that a tool is available that allows quick and easy development of reliable custom installers. Give InstallAnywhere a try. It could help make your task just a bit easier.

About the Author

Edward Zebrowski is a technical writer based in the Orlando, FL area. Ed runs his own Web development company, ZebraWeb, and can be reached on the Net at zebra@rock-n-roll.com

Browsing the JDBC API

Implement functional requirements by extending the core classes

by **Graham Harrison**

It is not easy to query the contents of a database without proprietary front end tools or a database-aware IDE. A database-aware toolkit should be able to connect to and work with a variety of databases (local and remote, application and corporate) without a shift in how we view the contents of different databases.

Java and JDBC allow the builder to abstract the viewing of the data from the implementation of the database and database queries that yield the data for viewing. A number of IDEs allow the builder to query the contents of a database as part of the database component integration , but using an IDE is an expensive option for someone who would like a simple tool for viewing and maintaining data in different databases.

This article presents a simple database browser that allows access to any database that supplies either an ODBC or JDBC driver, and acts as a simple but useful interface to the JDBC API. The browser can list the various types of table as well as the tables and table columns, and maintain data in those tables via SQL, which is entered free-form.

Most of this article addresses the design and build of the browser. The aim is to produce a complete tool written in 100% Java 1.1 (AWT and JDBC) with no third party add-ons, and which is scalable enough to allow a new presentation layer to be bolted on. It is also reasonable to expect browser components to behave as middleware components on a multi-tier platform so that the browser can be imported as a Java Bean™ into any Bean-complaint IDE. This will allow the 'Data Browser Bean' to interact with GUI and non-GUI components in a more ambitious distribution of functionality on a broader corporate platform. Although the browser does not implement the Bean interface, this is a small task for those interested in pursuing this extension.

Functionality

Essentially, the browser is a front end to the JDBC API. Based on practical requirements, the current version interfaces to a subset of the more important JDBC methods. The current user interface implements a set of three tabs using the Java CardLayout class:

- Query capture and display
- Table and column listing
- Connection detail

The user clicks on a button and the relevant tab detail is displayed.

Query Capture and Display

The user enters an SQL query which is sent to the database for execution. The query can be any SQL which the JDBC/ODBC driver and database supports; for example, select, insert, update, delete. The user can limit the number of rows displayed from a query.

Figure 1 shows a query entered into the query window; when the 'Go' button is clicked, the query is executed and the results displayed in the Panel above. The user can limit the number of rows returned from the ResultSet by entering a value in the 'Row Count' text box.

Note that the projection name (i.e., the name defined in the query), not the table column name, is displayed. If no projection

name were nominated, the table column name would be displayed.

Queries that are not select statements (e.g., Update, Insert) will return an update count, and a confirmatory message is appended to the panel display. Figure 2 shows the update message when the update query is executed.

Table and Column Listing

The user can select a qualified list of tables, including Catalogs, Schema and Table Types. From a list of tables, select the column detail for a specific table. Various

types of tables can be displayed; the column detail is the complete column specification returned by the JDBC API.

Connection Detail

The user can select the driver and database URL to connect with. The default is the JDBC/ODBC bridge supplied in JDK 1.1, and the jdbc:odbc protocol. Any valid driver can be entered and the database URL can be remote.The user can display important meta data information about the database connection and driver information.

JDBC Drivers and the JDBC/ODBC Bridge

The browser will accept the class name of any JDBC driver you have, but defaults to sun.jdbc.odbc.JdbcOdbcDriver, which is the JDBC/ODBC bridge supplied with JDK 1.1.

JDBC/Net drivers enable a connection from the client, thru a JDBC driver, to a comms server which then connects to a database server on a remote host. This is required when the JDBC driver does not inherently support the protocol required between the client and the server directly. The local comms server will handle the protocol of database requests and, in some cases, will make transparent the different versions of products on multiple platforms. Generally, a name service is provided on all cooperating hosts to direct the clients to host ports where the service is listening. For example, you could use JDBC to connect to an Ingres/NET comms server, which then establishes a connection to a remote comms server, which in turn connects to the remote database server, and a virtual circuit is then established. In the process, the local and remote name servers are queried to find the host/port addresses of the cooperating services.

JDBC drivers can also handle the communication protocol directly. In this case, the JDBC driver connects directly to the database server, local or remote.

For those data sources that do not have a JDBC driver, the JDBC/ODBC bridge allows an ODBC data source to be interrogated by the JDBC API. The bridge software converts the JDBC calls into ODBC calls.

The Database class in the Database Browser handles the database connection. The User Interface passes an instruction to the database via the protocol handler. The connectDB method uses the instruction operands as arguments to the JDBC API.

To use the ODBC data sources, you need to create a new System DSN from the Windows Control Panel. The name you assign to the database is the name you enter into the Database URL field in the Connection Tab. For example, the screenshots use the NorthWestTrade name, which maps down to the nwind.mdb Access database provided by Microsoft.

Interfaces	Comments
DBProtocol	Specifies protocol operators and operand offsets. Classes which implement this interface are sharing protocol constants.
DBConstants	Specifies application constants. Classes which implement this interface are sharing application wide constants.
DBHandle	Allows one component to refer to another. Classes which implement this interface are implementing a reference to the database object.
LogComponent	Allows one component to write to another. A class that implements this method must supply the code to write to a log device.

Table 1: Data Browser interfaces

The driver is loaded using the class.forname() method call. The getConnection() method call establishes a link to the database through the loaded driver. Once a link is created, a Statement object is created which is the handle for executing queries as in Listing 1.

Components (Interfaces and Classes)

The various components of the data browser are shown in Figure 5 and summarized in Tables 1 and 2.

Design and Build

To support a component approach, the design of the browser has the following design constraints which are explained in more detail in the following section.

- Separation of presentation layer from database layer
- Use of interfaces to support component integration
- Implementation of Observer/Observable from a model/view design pattern
- Protocol handling with the use of a protocol client and server
- Tight coupling of component to event using anonymous classes

Classes	Interfaces	Comments
DataBase	DBConstants DBProtocol	Data server. Handles the JDBC interface, including connection and API
DataBrowser	DBProtocol DBHandle DBConstants	User Interface. Manages an interface to the protocol client to ask the server to execute database requests
ProtocolClient	DBProtocol	Package a protocol instruction and dispatch it to the protocol server
ProtocolServer	DBProtocol	Interpret a protocol instruction and send a message to the Database to execute the operand
ProtocolData	DBProtocol	Create and store protocol instructions
LogDevice	LogComponent	Implements the write to a log device provided by the Data Browser. In the example, SQLLOG is a wrapper class for this output.

Table 2: Data Browser classes

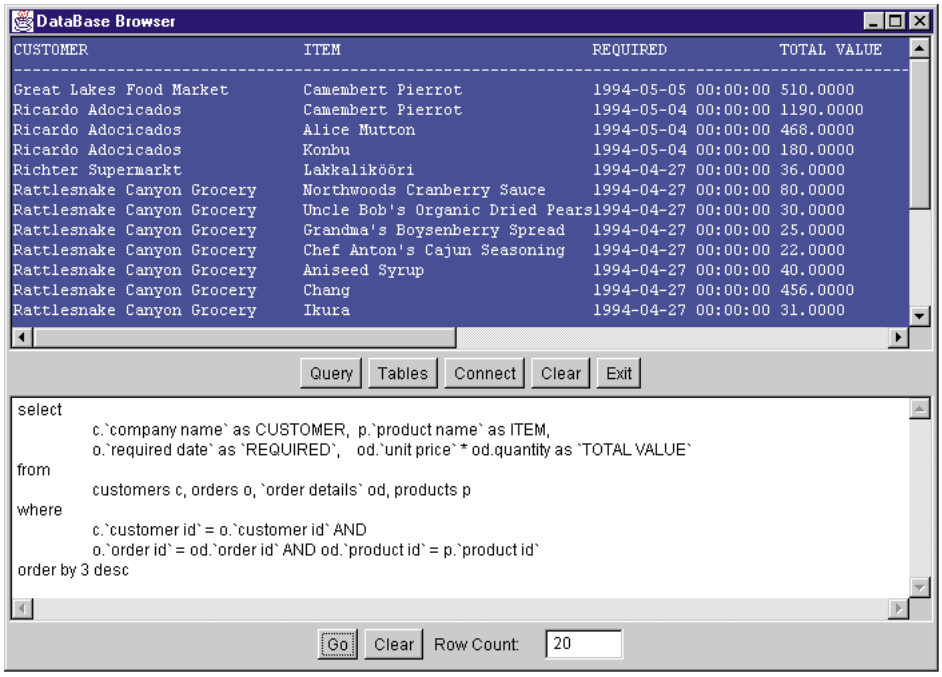


Figure 1: Select customers and their order items

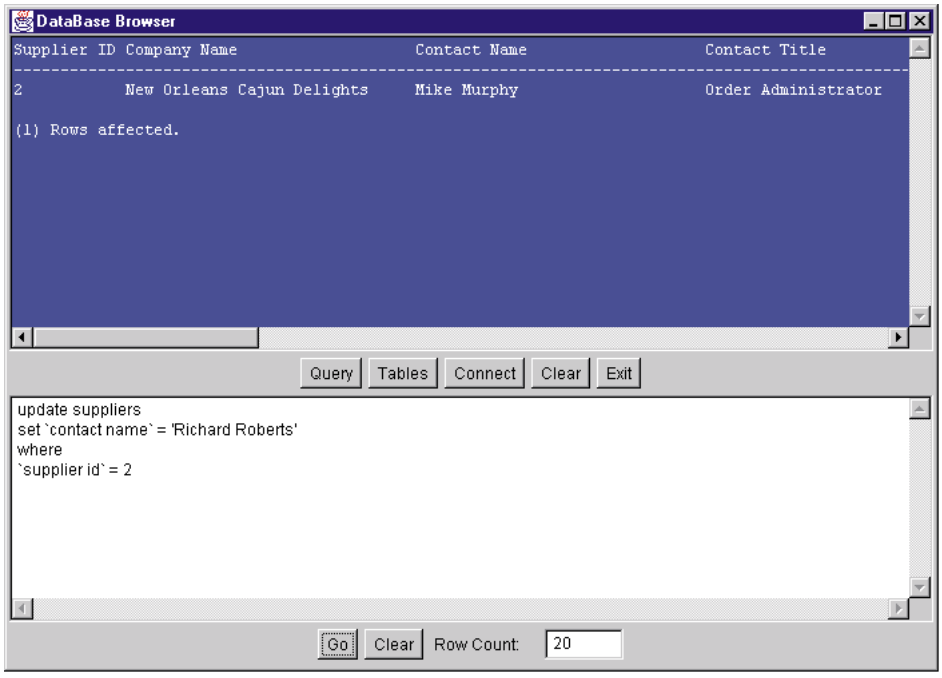


Figure 2: Update contact name for a supplier

- Strict use of core Java 1.1 AWT classes, including a CardLayout for a Tab Panel

Builders unfamiliar with anonymous classes should find the examples here straightforward.

Separation of Presentation Layer from Database Layer

The user interface classes are separate from the database classes. The user interface communicates with the service classes (DataBase) via the protocol handler. The database performs the requested service and obtains a handle to two items: a results component to display JDBC API results and a log device to contain standard error output.

There are three ways to approach the handling of query results from the database:

a. the database returns query results as Java properties, e.g. Vectors and arrays
b. The database class writes the results to a device owned by the presentation layer.
c. The database class invokes a method provided by the presentation layer.

In the actual implementation:

a. is used to store meta values from the database in a class called ResultsInfo.
b. is used to write the API results. This includes the results of queries and meta-data API calls.
c. is used for standard error output; since standard error is assumed to be a property of the user interface, the user interface must provide a class that implements the LogDevice interface.

Use of a log device that is visible to the database class yields a number of benefits. Exceptions can be handled and reported within the database service itself. If this were not the case, then either the protocol handler or the user interface would have to catch the exceptions thrown by the database. This reduces the granularity of error handling, leading to a lack of flexibility. Some of the SQL Exceptions thrown are not malevolent; a JDBC API method may not be supported by the ODBC driver (e.g., getSchemas()). In this case, we want to continue in the database server, not return to the client. We also want to report this fact to standard error.

Use of Interfaces to Support Component Integration

A small number of interfaces are used to ensure that a design thread is followed through in each relevant component. The interfaces perform two functions:

- Specification of constant values so that interacting components can share them (interfaces DBConstants and DBProtocol)
- Allow method signatures to be specified so that one component can invoke a method whose detail is completed by another component (interface LogComponent)

Implementation of Observer/Observable from a Model/View Design Pattern

The User Interface (DataBrowser class) is an observer of the Data (DataBase class). On a client/server platform with multiple client access to the data, if one client updates the data then the other clients may need to be informed and act accordingly. This means that the user interface is an observer of the data; the data is 'Observable'. Although the implementation described here does nothing with this paradigm, for extensibility the user interface has been defined as an observer and the database as observed. The Observer 'update()' method is a no-op method. The Observed 'notify()' method is similarly a no-op method. If the DataBase requires all registered observers to update their state based on a change to the database, then the DataBase should notify() all observers; the observers (the user interface) should then execute the update() method. Note that the data passed is still bound to the protocol, so a ProtocolData is used (see Listing 2).

ad

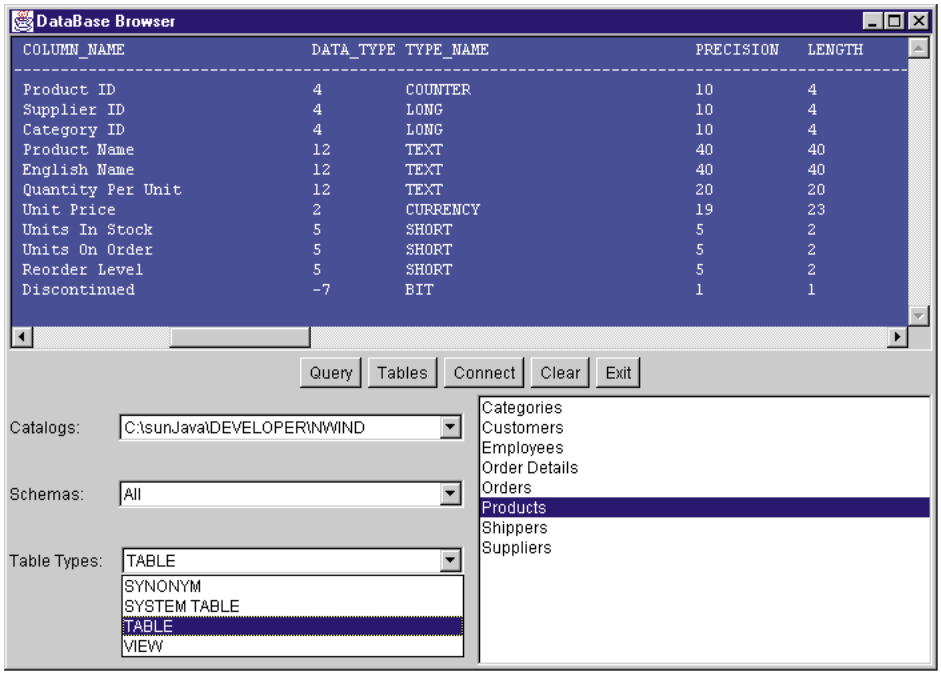


Figure 3: Column detail of the Products Table

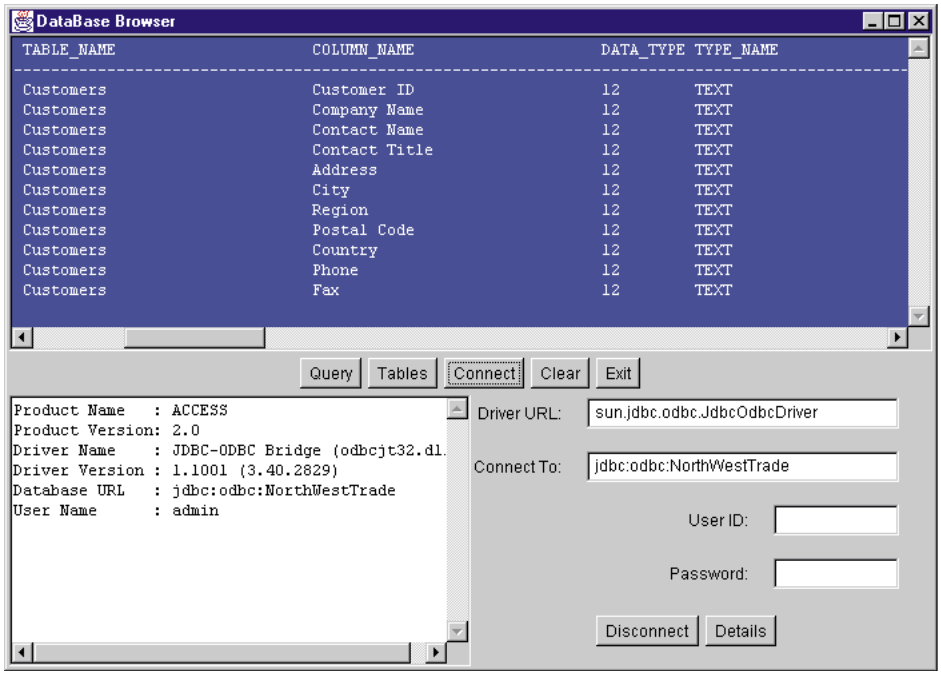


Figure 4: Connection tab after the Details Button is clicked

Protocol Handling with the use of a Protocol Client and Server

Most client/server and n-tier architectures implement some form of protocol handling between the clients and the servers (either database servers or middle-tier application servers which communicate with the persistent storage servers). This allows the clients to abstract the operational functionality from the implementation of that functionality. On a distributed client/server platform, a communications server is usually required to take requests from the client and pass them on to the comms server on the server machine, which then passes the request on to a database server. The requests are formalized into a protocol which consists of an instruction with data or, more formally, an operator with operands. Client components which interact with a service on a remote host need to understand how to package up the instructions, or protocol data, into a suitable form.

The data browser uses a 'protocol client' class to package up a protocol instruction into a 'protocol data' class which then forwards this instruction to the 'protocol server' class to interpret, the interpretation being a call to a DataBase method to execute a JDBC method call.

What seems like overengineering a simple procedure yields dividends when a host implements a different service. The client will then require access to a different protocol client to connect to this service, the simple cost being the implementation of a new protocol interface to parameterize new service instructions instead of hardcoding the service calls manually.

The ProtocolClient class contains a method to package up operand values and send a protocol instruction to the protocol server. The example in Listing 3 takes the query entered by the user, and the row limit, as operands, and sends an execute query instruction to the ProtocolServer object.

The ProtocolServer class in Listing 4 contains a method that unpacks these operands and executes the instruction opcode. The object passed between the protocol client and server is an instance of ProtocolData class. Listing 5 shows the ProtocolData class handling all the operand/operator capture.

The ProtocolData object is a Singleton object; i.e., one instance of the class is created, regardless of how many times the constructor is called:

```
private static ProtocolData pd;
public ProtocolData() {
    if (pd == null) {
        pd = this;
    }
}
```

The ProtocolClient and ProtocolServer classes are never instantiated; the relevant variables and methods are declared as static so they are referenced at class level. There is no need to instantiate them.

A distributed, multi-tier treatment of the protocol handling would use RMI, in which the call to the serviceDBRequest() methods is an RMI call. Without RMI, the ProtocolData object must be serializable, with the protocol client and server connected over sockets.

Tight Coupling of Component to Event using Anonymous Classes

Java 1.1 introduced both event delegation and anonymous classes. Event delegation allows events generated from one component – e.g. a button – to be handled by another component. Anonymous classes allow the event handling code to be specified when registering the event listener for that component without the need to define and instantiate another class, reducing design and build complexity. The 'Anonymous' tag describes a class that has no name; as far as the builder is concerned, it is in-line event handling code. When the code is compiled, a class file is generated which contains the event handling code and the class is named after the containing class with a '$' and a class number suffixed to it.

ADVERTISER INDEX

Advertiser		Page
3D Graphics		27
www.threedgraphics.com	310 553-3313	
Activerse		21
www.activerse.com	512 708-1255	
Bristol Technology		63
www.bristol.com	203 438-6969	
DCI		68, 69
www.DCIexpo.com/Internet	508 470-3880	
Greenbrier & Russel		19
www.gr.com/java	800 453-0347	
Imperial		77
www.Imperial.com	415 688-0200	
Installshield		31
www.installshield.com	800 250-2191	
Intuitive		50
www.intuitivesystems.com	408 245-8540	
JavaWorld		83
www.javaworld.com	415 267-4527	
KL Group Inc.		84
www.klg.com	800 663-4723	
McGraw-Hill		41
www.mcgrawhill.com	800 227-0900	
Marimba		63
www.marimba.com/download	415 328-JAVA	

Advertiser		Page
Mecklermedia		81
www.iworld.com	800 500-1959	
MindQ		35
www.mindq.com	800 847-0904	
Net-Developer '98		59
www.net-developer.com	612 368-7227	
Net Guru		67
www.ngt.com	800 know.ngt	
Net Guru		77
www.ngt.com	800 know.ngt	
Object Matter		56
www.objectmatter.com	305 718-9109	
Petronio Technology Group		33
www.petronio.com	781 7788-2000	
Phaos		23
www.Phaos.com	212 229-1450	
PreEmptive Solutions, Inc.		25
www.preemptive.com	216 732-5895	
Progress Software		29
www.progress.com	617 280-4000	
ProtoView		19
www.protoview.com	609 655-5000	
Sales Vision		15
www.salesvision.com	704 567-9111	

Advertiser		Page
SofTech Computer Systems		33
www.scscompany.com	814 696-3715	
Software Development West		57
www.sd98.com	800 411-8826	
Stingray Software Inc.		2
www.stingsoft.com	800 924-4223	
SunTest		67
www.suntest.com	415 336-2005	
SuperCede		6
www.asymetrix.com	800 448-6543	
Sybex Books		37
www.sybex.com	510 523-8233	
Symantec		3
cafe.symantec.com	800 453-1059 ext. 9NE5	
SYS-CON Publications		61
www.sys.con.com	914 735-1900	
Thought, Inc.		29
www.thought.com	415 836-9199	
Waite Group Press		43
www.waite.com	800 368-9369	
Zero G. Software		54
www.zerog.com	415 512-7771	

1/4 Ad

1/4 Ad

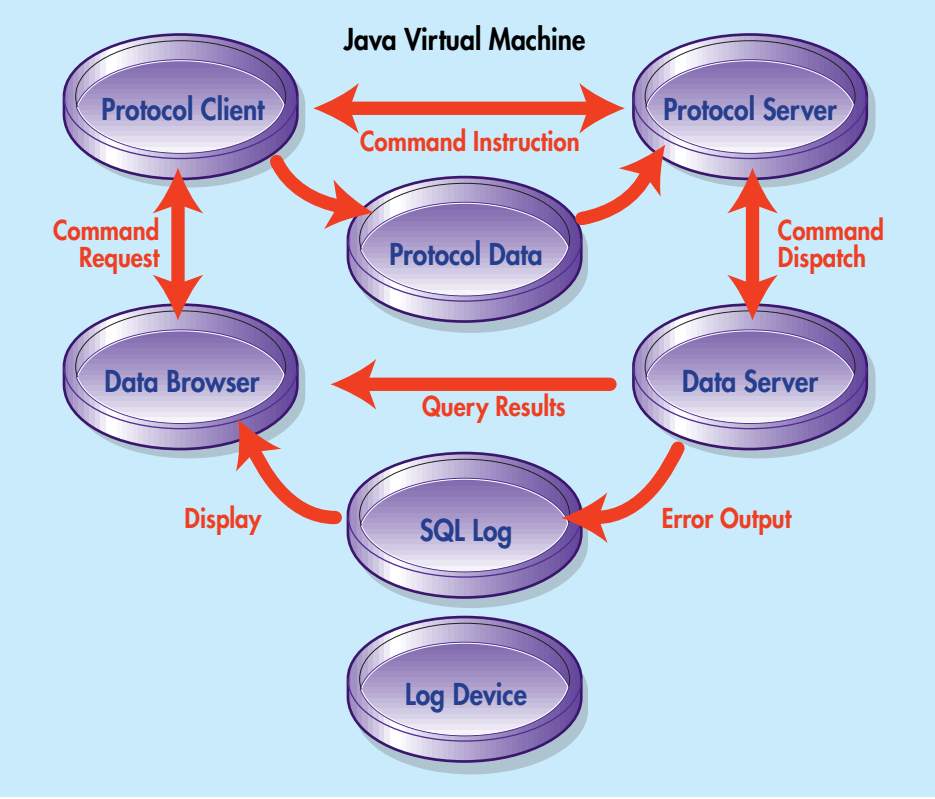


Figure 5: Cooperating Data Browser Objects

The user interface makes use of anonymous classes. Listing 6 shows how, when an entry in the tableList is clicked, a request is sent for the table columns to be displayed. Instead of adding a class which implements the ItemListener interface, Java will create a class anonymously at compilation and add it as a listener to the tableList component when the code above is executed.

Listing 7 shows how to display the Query Window when the Query button is pressed.

Strict Use of Core Java 1.1 AWT Classes

Although several widget libraries exist which support grid and tree controls, the browser does not use any and instead uses the core AWT components and layout managers. This reduces some of the complexity between the user interface and the protocol handling and simplifies the design, but makes for a less attractive interface and more coding.

In a prototyping context, the ideal would be to use a Bean-compliant IDE and Bean-compliant widgets, of which there are many. However, the design constraints dictate the core functional requirement that a user interface be placed on top of the JDBC API. Some would argue that the user interface is the most disposable part of an application; this is true if the parts are built quickly and cheaply.

Evaluation of GUI components is outside the scope of this exercise, so a decision was made to use the core AWT components, using a CardLayout to implement a form of tabbed panelling. Good use is made of the CardLayout component, which implements tab-like functionality. Here, the user clicks a button (Query, Tables, Connect) and a different card, or 'Tab', is displayed. The contents of the tab persist between displays so that, for example, the list of tables is still displayed when entering and returning from the Query tab.

When the user clicks one of the central buttons, the relevant tab is displayed as follows (using an anonymous class):

```
connectButton.addActionListener(
    new ActionListener() {
        public void actionPerformed(
            ActionEvent e){
                tabOptions.show(
                    panel4,
                    connectButton.get-
Label());
}});
```

Panel panel4 is a panel that has been added to the tabOptions CardLayout component and has a tag that is equivalent to the text of the button which fires the action event. The Connect tab is shown when the application starts up by setting the default display property as follows:

```
tabOptions.first(panel4);
tabOptions.show(panel4, connectButton.getLa-
bel());
```

Extensions and Conclusion

The Data Browser allows us to view the JDBC API in a highly flexible way without compromising portability. With the source code provided, this article will enable you to use and extend the browser, learn how to work with layout managers to achieve a reasonably sophisticated effect without third party widget class calls and, centrally, learn how to connect to a database and use JDBC method calls.

Clearly, not all of the API has been implemented; interested readers can include extra method calls using popup menus against different menu options; e.g. metadata, statements, connection details, etc. It is a moderately simple task to hook another user interface to the protocol and database classes.

One enhancement would be to attach a

Listing 1: Connect to Database using Protocol Operands

```
// Connect to database
public void connectDB ( String []  operand )
                    throws Exception {
    if (operand[DBP_DRIVER_ARG] != null ) {
      class.forName( operand[DBP_DRIVER_ARG]);
    }
    if (operand[DBP_USERID_ARG] ==  null) {
       session =
       DriverManager.getConnection(
         operand[DBP_URL_ARG]);
    }
    else {
       session =
           DriverManager.getConnection(
      operand[DBP_URL_ARG],
      operand[DBP_USERID_ARG],
      operand[DBP_PASSWORD_ARG]  );
    }
    query =  session.createStatement();
}
```

Listing 2: The User Interface is notified of Observable change with a ProtocolData Object.

```
class DataBrowser
         extends Frame
         implements DBProtocol,
```

grid component to the user interface. There are several excellent grid components available for 1.0 and 1.1 Java; Swing grids are available for 1.2.

The next enhancement would be to generate screens dynamically based on the columns of tables that we want to update. For example, we may select a database and a table, then nominate columns for a form that would be used to qualify queries for insert, update and delete queries. Remember that the browser is not tied to one database design so the forms must be created dynamically.

Network distribution of components can be achieved with little modification to the code through an RMI implementation, with the User Interface and Protocol Client on one host and the Protocol Server and DataBase on another host. All four components could be on four different hosts. One drawback in multi-tier hosting of service objects is the serialization and network transport of marshalled objects. Java 1.1 introduces object serialization; it is very easy to implement this feature without using RMI. Essentially, the ProtocolData instance is serialized and passed over a TCP/IP socket connection to a listening protocol server. Since both client and server are Java, you can use RMI, which implements serialization underneath the RMI protocol.

For an implementation where many clients require access to the database, the main requirement is that the Database class makes all methods 'synchronized' so that there is no contention when updating or inserting data. Where many clients require the protocol handler to perform a service, the ProtocolData object must not be a Singleton object, but be instantiated for every client that requires a protocol instruction to be formatted. The ProtocolClient and ProtocolServer classes would need some adjustment to enable the client/database handles to retain their identity for each client. For example, the ProtocolClient could be instantiated for every user interface connection, the variable data being a reference to the main UI object (the methods remain static); alternatively, a static array could contain a list of references to controlling user-interface objects.

In summary, when we relocate cooperating objects from the same virtual machine, we need to implement RMI or sockets while retaining the component functionality. Most functional requirements you explore can be implemented by extending the core classes in this way.

About the Author

Graham Harrison is a Senior Consultant with Informix Software and a Sun Certified Java Programmer. He can be contacted at gpharrison@compuserve.com

gpharrison@compuserve.com


```
                            DBConstants,
                            DBHandle,
                            ItemListener,
                            Observer {
…

        public void update(  Observable o,
                                        Object p) {
                if !(p instanceof ProtocolData)){
                     System.out.println("Error in
                                Observers.");
                     System.exit(1);
                }
                // do something as directed by
                // observable object with detail
                // in the protocol, e.g. requery
      }}
```

Listing 3: The Protocol Client creates a Protocol Instruction.

```
public static void  tellProtocolClient( byte opcode )
                                throws Exception {
    ProtocolData protocolData =
            ProtocolData.getProtocolData();
    protocolData.setOpcode(opcode);
    try {
        switch (opcode) {
            case DBP_EXECUTE_QUERY: {
                   // the TextArea is the operand value
                protocolData.setOperand(
                            ui.queryWindow.getText(),
                            DBP_SQLTEXT_ARG);
                protocolData.setOperand(
                           ui.rowCount.getText(),
                           DBP_ROWCOUNT_ARG);
                ProtocolServer.serviceDBRequest(
                            protocolData);
                break;
 }
…
}
```

Listing 4: Protocol Server Instruction Handling.

```
// protocol handler
public static Object serviceDBRequest( ProtocolData p )
                                    throws Exception {
    switch (p.getOpcode()) {
        case DBP_EXECUTE_QUERY: {
            db.execSQL(
                 p.getOperand(DBP_SQLTEXT_ARG),
                 p.getOperand(DBP_ROWCOUNT_ARG));
 return db.getResults();
        }
…
}
```

Listing 5: ProtocolData Class Operand Capture

```
public void setOpcode( int opcode ) {
    this.opcode = opcode;
}
public void setOperand( String operand, int index  ) {
    this.operand[index] = operand;
}
```

Listing 6: Anonymous Class Example One

```
tableList.addItemListener(
    new ItemListener() {
        public void itemStateChanged(
                                ItemEvent e ) {
            Integer index =  (Integer)e.getItem();
            tableName = tableList.getItem(index.intValue());
            try {
                ProtocolClient.tellProtocolClient(
                                        DBP_COLUMNS );
            } catch (Exception ea) {
                 ea.printStackTrace();
            }
       } });
```

Listing 7: Anonymous Class Example Two.

```
queryButton.addActionListener(
        new ActionListener() {
            public void actionPerformed(
                                ActionEvent e ){
                tabOptions.show(
                           panel4,
                           queryButton.getLabel());
```

Implementing Callback-Style Support for Java's AWT

Allow your application to better survive any future AWT paradigm shift

by **Daniel Dee**

What is a Callback?

A callback is a mechanism by which the user's action on a software application's graphical user interface (GUI) is connected to the code implementing the application's response to this action. It is a familiar concept to X Toolkit and Motif programmers. In reality, the actual callback is the part of the callback mechanism that implements the application response. In this sense, a callback is also known as a command in object-oriented circles. This article describes how to implement support for a Java version of the X Toolkit/Motif-style callback for AWT.

Advantages of Using Callbacks

There are many advantages of using callbacks instead of standard AWT event handling:

- It clearly separates the application's GUI code from the code implementing the application's response to any user interaction with the GUI.
- For those programming in Java 1.0, there is no need to subclass an AWT component to implement an action, thus preventing subclass proliferation.
- For those programming in JDK 1.1, Listeners instances are centralized in one location.
- More importantly, Java version-dependent code is localized in one class. Thus, application code written using the callback mechanism achieves Java version independence. In other words, an application may be ported easily from JDK 1.0 to 1.1 without rewriting a single event handling code.

How Callback Works

Figure 1 shows the workings of the callback mechanism. It shows the relationship of the event originator, the event handler and the callback. The event handler is the centralized location where all AWT events are delivered, and it decides whether any application response code is interested in processing this event. This is implemented in the CallbackList class. The AWT component is where the event originated has to be extended or subclassed to hand off events to the centralized event handler rather than processing the event itself. The programmer supplies the application response callback code by extending the Callbackable class, and registers this code with the extended AWT component.

The Callbackable Class

Callbackable is implemented as an abstract class to enforce a uniform structure to all callback code. The callback code is where the programmer implements the application's response to an event. This is necessary because the CallbackList expects a specific method in a registered callback; specifically, public abstract void boolean activate(Event e). Information related to the component and the event that triggers this code can be retrieved from the Event object that is passed as the parameter. The actual implementation of the Callbackable interface is very simple and is shown in Listing 1.

The CallbackList Class

The CallbackList maintains a list of registered instances of Callbackable. This is also where the majority of the code for handling the AWT differences between Java 1.0 and 1.1 is located. In 1.0, the CallbackList is fairly simple. It extends the Vector class and keeps a list of registered instances of Callbackable. In JDK 1.1, this class becomes a little bit more complicated as it also has to act as the Listener of all AWT components. Listing 2 shows the complete source code for the Java 1.0 version of CallbackList, and Listing 3 shows the same for Java 1.1. When the programmer registers an instance of a concrete subclass of Callbackable, it gets added to the CallbackList. Multiple instances of Callbackable may be registered for a single AWT component in CallbackList. When an interaction occurs with an AWT component, the event is passed to a specific instance of the CallbackList for that AWT component, which goes through its Vector of registered instances of Callbackable and activates each one in turn.

Adding Support for Callbackable in AWT

A user action reaches an instance of an AWT component as an event. This event must be passed on to the CallbackList where it can be redirected to the appropriate Callbackable. To do this, an AWT component must be extended. Listings 4 and 5 show the extended AWT Button as an example for Java 1.0 and 1.1 respectively, but other AWT components can be extended the same way. In implementing the extension to AWT Button, one has to decide what events or compound events are important and a CallbackList instantiated to represent each. In this case, it will be the activate (mouse click), arm (mouse down) and disarm (mouse up) events. So, the activateCallbackList, armCallbackList and disarmCallbackList are created for it. The programmer registers an instance of a Callbackable using the addCallback method. AddCallback simply adds a Callbackable to the appropriate CallbackList. Instead of requiring the programmer to subclass Button and write a new handleEvent code for each button that was instantiated, the programmer inserts the operation code in the subclass of Callbackable itself. This enforces the division between GUI code and operational code and makes the code more reliable and reusable.

Using Callback

Listing 6 shows a sample program that uses the "enhanced" Button class. A concrete subclass of Callbackable is instantiated and added to an instance of the Button class. Note the use of a constructor and the set methods to pass extra information to the Callbackable. To find out which component triggered this callback, e.target is used. A single application will suffice

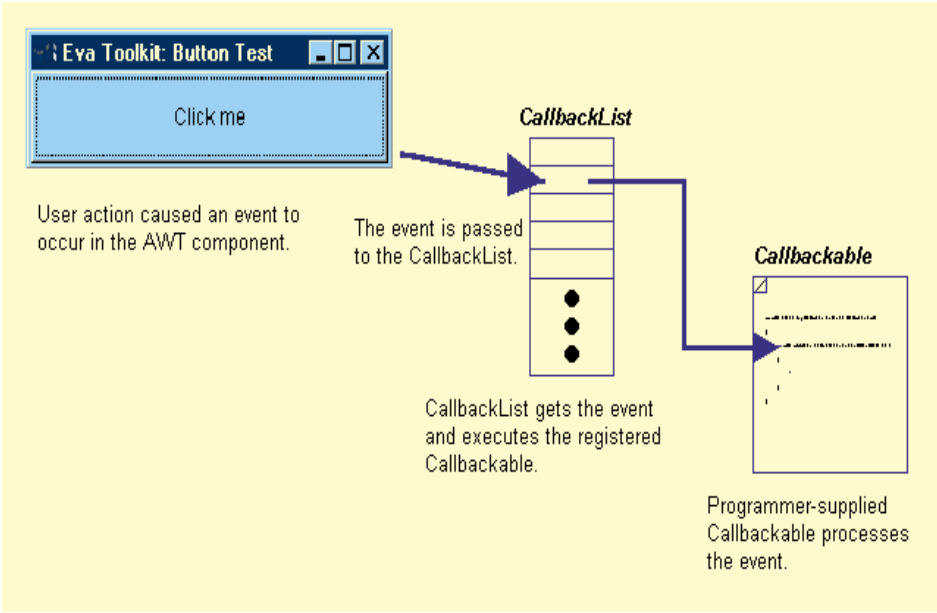


Figure 1: How Callback Works

for both Java 1.0 and 1.1 because code specific to the Java version has been encapsulated in the implementation of callback support.

Conclusion

In many ways, the callback mechanism is similar to listeners in Java 1.1, where the actual operational code is delegated to an instance of the Listener class. However, with listeners, an event specific method in an event source specific class is triggered for each specific event. For example, a mouse click will trigger the mouseClicked method in MouseListener. While with callbacks, there is only one class, Callbackable, and only one method, activate, to remember. In addition, implementing a callback layer over Java's own event handling allows one's application to better survive any future AWT paradigm shift.

The follow-up article to this will discuss how to enforce uniformity in the callback-extended AWT components by using the Widget interface class.

Download Source Code

Source code for this article can be downloaded free from http://www.wigitek.com. A fully implemented version containing extended AWT code can also be purchased from Wigitek Corporation at the same Web site.

About the Author

Daniel Dee has more than 10 years of experience working in the development of GUI software toolkits, starting with X Windows and then Java, since their inception. He is currently the president of Wigitek Corporation, a company providing software tools and consulting services for the development Java-based data-driven dynamic graphics software. He received an MS degree in Computer System Engineering from the University of Massachusetts. Daniel can be reached at daniel@wigitek.com

daniel@wigitek.com

Listing 1.

```
/**
 * Copyright (c) 1997 Daniel Dee
 * Description:
   Package contains common classes for the ViviGraphics Widget
   Toolkit - a callback-based toolkit.
   Originally, part of the Eva Toolkit - the prototype
   implementation.
 */
package com.wigitek.vivigraphics.widget.common;

import java.awt.Event;

/**
 * This class implements the callback mechanism that is required
   to support the Widget interface. You will typically subclass
   this to implement your own activate method in order to perform
   functions specific to events that trigger the callback.
 * @version $Revision$
 * @author Originally written by Daniel Dee, 3/17/97
 * @author Last updated by $Author$, $Date$
 */
public abstract class Callbackable extends Object
{
    /**
     * Performs functions specific to the event that triggers this
       callback. By default, it prints information about the acti-
vating
       object when this callback was registered and the activating
event.
       You should normally override this method to perform func-
tion
       specific to your application.
     * @param evt        the event that triggers this callback
     */
    public abstract boolean activate( Event e );
}
```

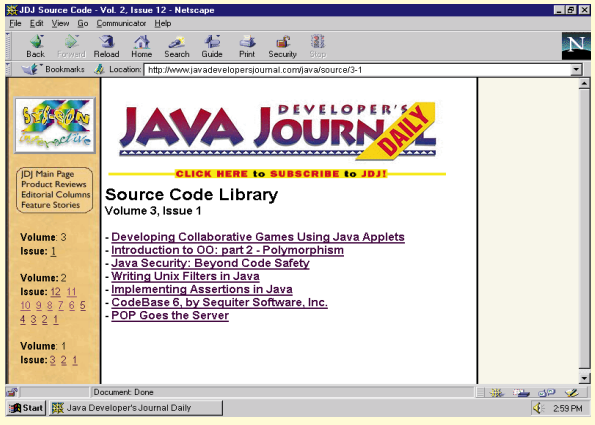
Listing 2.

```
/**
 * Copyright (c) 1997 Daniel Dee
 * Description:
   Package contains common classes for the Vivigraphics Widget
   Toolkit - a callback-based callback-based toolkit.
   Originally, part of the Eva Toolkit - the prototype
   implementation.
 */
package com.wigitek.vivigraphics.widget.gui;

import java.awt.Event;
import java.util.Vector;
import java.io.IOException;
import com.wigitek.vivigraphics.widget.common.Callbackable;

/**
 * This class chains callbacks associated with a single callback
type
   together.
 * @version $Revision$
 * @author Originally written by Daniel Dee, 3/17/97
 * @author Last updated by $Author$, $Date$
 */
public class CallbackList extends Vector
{
    /**
     * Constructs a CallbackList. Creates a Vector with 100 ini-
tial elements
       and a increment size of 100.
     */
```

```
public CallbackList()
{
    super(100, 100);
}

/**
 * Removes a Callbackable from the CallbackList.
 * @param cb the callback
 */
public boolean remove( Callbackable cb )
{
    boolean returnValue = false;

    for( int i=0; i < size(); i++ )
    {
        Callbackable _cb = (Callbackable)elementAt(i);
        if( _cb == cb )
        {
            removeElementAt(i);
            returnValue = true;
        }
    }
    return returnValue;
}

/**
 * Removes a CallbackClientPair at the indexed position.
 * @param index the position of the CallbackClientPair in the
                CallbackList.
 */
public boolean removeAt( int index )
{
    try
    {
        removeElementAt( index );
        return true;
    }
    catch( ArrayIndexOutOfBoundsException e )
    {
        return false;
    }
}

/**
 * Adds a callback to a chain.
 * @param callback     the callback
 * @return the Callbackable
 */
public Callbackable add( Callbackable cb )
{
if( cb == null )
 cb = defaultCB;
    addElement( cb );
    return cb;
}

/**
 * Calls all the callbacks in this chain.
 * @param evt the event that triggers this callbacklist
 * @return true if event has been processed by the callback
           and no further processing is necessary; false
           if further processing from the activating object
           is required.
 */
public boolean activate( Event e )
{
    boolean returnValue = true;

    for( int i=0; i < size(); i++ )
    {
        Callbackable cb = (Callbackable)elementAt(i);
        returnValue = cb.activate( e );
    }

    return returnValue;
}

    private CallbackListDefaultCallbackable defaultCB =
       new CallbackListDefaultCallbackable();
}

/**
 * Default Callbackable.
 */
class CallbackListDefaultCallbackable extends Callbackable
{
    /**
     * Prints information about the activating
       object.
     * @param evt        the event that triggers this callback
     */
    public boolean activate( Event evt )
    {
        System.out.println( "Calling object is " +
evt.target.toString() + "." );
        System.out.println( "Activating event is " +
evt.toString() + "." );
        return true;
    }
}
```

Don't Type it... Download it!

Access the remainder of the source code for this article at JavaDevelopersJournal.com



Listing 3.

```
/**
 * Copyright (c) 1997 Daniel Dee
 * Description:
   Package contains common classes for the Vivigraphics Widget
   Toolkit - a callback-based callback-based toolkit.
   Originally, part of the Eva Toolkit - the prototype
   implementation.
 */
package com.wigitek.vivigraphics.widget.gui;

import java.awt.Event;
import java.awt.AWTEvent;
import java.util.Vector;
import java.io.IOException;
import java.awt.event.*;
import com.wigitek.vivigraphics.widget.common.Callbackable;

/**
 * This class chains callbacks associated with a single callback
type
   together.
 * @version $Revision$
 * @author Originally written by Daniel Dee, 3/17/97
 * @author Last updated by $Author$, $Date$
 */
public class CallbackList extends Vector
                        implements ActionListener,
                                   AdjustmentListener,
                                   ComponentListener,
                                   ContainerListener,
                                   FocusListener,
                                   ItemListener,
                                   KeyListener,
                                   MouseListener,
                                   MouseMotionListener,
                                   TextListener,
                                   WindowListener
{
    /**
     * Constructs a CallbackList. Creates a Vector with 100 ini-
tial elements
       and a increment size of 100.
     */
    public CallbackList()
    {
        super(100, 100);
    }

    /**
     * This method is called when an action event occurs inside
the
       source object. The corresponding callbacks added
       by the user are activated.
     * @param evt the event
     */
    public void actionPerformed(ActionEvent evt)
    {
    Event e = new Event( evt.getSource(), evt.getID(), evt );
        activate(e);
    }

 // TO BE IMPLEMENTED
    public void adjustmentValueChanged(AdjustmentEvent evt) {}
    public void componentResized(ComponentEvent evt) {}
    public void componentMoved(ComponentEvent evt) {}
    public void componentShown(ComponentEvent evt) {}
    public void componentHidden(ComponentEvent evt) {}
    public void componentAdded(ContainerEvent evt) {}
    public void componentRemoved(ContainerEvent evt) {}
    public void focusGained(FocusEvent evt) {}
    public void focusLost(FocusEvent evt) {}
    public void itemStateChanged(ItemEvent evt) {}
    public void keyTyped(KeyEvent evt) {}
    public void keyPressed(KeyEvent evt) {}
    public void keyReleased(KeyEvent evt) {}
    public void mouseClicked(MouseEvent evt) {}
    public void mousePressed(MouseEvent evt) {}
    public void mouseReleased(MouseEvent evt) {}
    public void mouseEntered(MouseEvent evt) {}
    public void mouseExited(MouseEvent evt) {}
    public void mouseDragged(MouseEvent evt) {}
    public void mouseMoved(MouseEvent evt) {}
    public void textValueChanged(TextEvent evt) {}
    public void windowOpened(WindowEvent evt) {}
    public void windowClosing(WindowEvent evt) {}
    public void windowClosed(WindowEvent evt) {}
    public void windowIconified(WindowEvent evt) {}
    public void windowDeiconified(WindowEvent evt) {}
    public void windowActivated(WindowEvent evt) {}
    public void windowDeactivated(WindowEvent evt) {}

    /**
     * Removes a Callbackable from the CallbackList.
     * @param cb the callback
     */
    public boolean remove( Callbackable cb )
    {
        boolean returnValue = false;

        for( int i=0; i < size(); i++ )
        {
            Callbackable _cb = (Callbackable)elementAt(i);
            if( _cb == cb )
            {
                removeElementAt(i);
                returnValue = true;
            }
        }
        return returnValue;
    }

    /**
     * Removes a CallbackClientPair at the indexed position.
     * @param index the position of the CallbackClientPair in the
                    CallbackList.
     */
    public boolean removeAt( int index )
    {
        try
        {
            removeElementAt( index );
            return true;
        }
        catch( ArrayIndexOutOfBoundsException e )
        {
            return false;
        }
    }

    /**
     * Adds a callback to a chain.
     * @param callback     the callback
     * @return the Callbackable
     */
    public Callbackable add( Callbackable cb )
    {
    if( cb == null )
     cb = defaultCB;
        addElement( cb );
```

```
        return cb;
    }

    /**
     * Calls all the callbacks in this chain.
     * @param evt the event that triggers this callbacklist
     * @return true if event has been processed by the callback
                and no further processing is necessary; false
                if further processing from the activating object
                is required.
     */
    public boolean activate( Event e )
    {
        boolean returnValue = true;

        for( int i=0; i < size(); i++ )
        {
            Callbackable cb = (Callbackable)elementAt(i);
            returnValue = cb.activate( e );
        }

        return returnValue;
    }

    private CallbackListDefaultCallbackable defaultCB =
       new CallbackListDefaultCallbackable();
}

/**
 * Default Callbackable.
 */
class CallbackListDefaultCallbackable extends Callbackable
{
    /**
     * Prints information about the activating
       object.
     * @param evt         the event that triggers this callback
     */
    public boolean activate( Event evt )
    {
        System.out.println( "Calling object is " +
evt.target.toString() + "." );
        System.out.println( "Activating event is " +
evt.toString() + "." );
        return true;
    }
}
```

Listing 4.

```
/**
 * Copyright (c) 1997 Daniel Dee
 * Description:
   Package contains common classes for the ViviGraphics Widget
   Toolkit - a callback-based toolkit.
   Originally, part of the Eva Toolkit - the prototype
   implementation.
 */
package com.wigitek.vivigraphics.widget.gui;

import java.awt.Event;
import java.lang.String;
import com.wigitek.vivigraphics.widget.common.Callbackable;
import com.wigitek.vivigraphics.widget.gui.CallbackList;

/**
 * This class extends the java.awt.Button class to support
   the callback mechanism by implementing the Widget interface.
 * @version $Revision$
 * @author Originally written by Daniel Dee, 2/21/97
 * @author Last updated by $Author$, $Date$
 * @see Widget
 */
public class Button extends java.awt.Button
{
    /**
     * Constructs a Button with no label and with name "unnamed".
     */
    public Button()
    {
        this( "", "unnamed" );
    }

    /**
     * Constructs a Button with a string label and with name
"unnamed".
     * @param label the specified label
     */
    public Button( String label )
    {
        this(label, "unnamed");
    }

    /**
     * Constructs a Button with a string label and with given
name.
     * @param label the specified label
     * @param name  the specified name
     */
    public Button( String label, String name )
    {
        super(label);
        this.name = name;
    }

    /**
     * Returns a String that represents the value of this Object.
       Overrides the method in java.lang.Object.
     * @return a String
     * @see Object
     */
    public String toString()
    {
        String string = super.toString();
        int length = string.length();
        return string.substring( 0, length-1 ) + ",name=" + name
+ "]";
    }

     /**
     * Registers a callback for the top level frame. Top level
frame
       currently recognizes only ACTIVATE_CALLBACK when the par-
ticular
       instance of the button is pushed. Unknown callbacks
       are ignored.
     * @param callbackName the type of callback to register
     * @param callback     the object of the Callbackable class
that will activated
                             when trigger by the callback type
given by callbackName
     * @see Callbackable
     */
    public void addCallback( String callbackName, Callbackable
callback )
    {
        if( callbackName.compareTo(ACTIVATE_CALLBACK) == 0 )
            activateCallbackList.add( callback );
    }

    /**
     * This method is called when an action event occurs inside
```

```
this
       Button. Depending on the event, the corresponding callbacks
added
       by the user are activated.
     * The method returns true to indicate that it has successful-
ly
       handled the action; or false if the event that triggered
       the action should be passed up to the component's parent.
     * @param evt the event
     * @return true if the event has been handled and no further
               action is necessary; false if the event is to be
               given to the component's parent.
     */
    public boolean action(Event evt, Object arg)
    {
        activateCallbackList.activate( evt );
        return true;
    }

    // Name of instance of this class.
    private String name;

    // Callback when button is pushed.
    private CallbackList activateCallbackList = new Callback-
List();
    public static final String ACTIVATE_CALLBACK = "activateCall-
back";
}
```

Listing 5.

```
/**
 * Copyright (c) 1997 Daniel Dee
 * Description:
   Package contains common classes for the ViviGraphics Widget
   Toolkit - a callback-based toolkit.
   Originally, part of the Eva Toolkit - the prototype
   implementation.
 */
package com.wigitek.vivigraphics.widget.gui;

import java.awt.Event;
import java.lang.String;
import com.wigitek.vivigraphics.widget.common.Callbackable;
import com.wigitek.vivigraphics.widget.gui.CallbackList;

/**
 * This class extends the java.awt.Button class to support
   the callback mechanism by implementing the Widget interface.
 * @version $Revision$
 * @author Originally written by Daniel Dee, 2/21/97
 * @author Last updated by $Author$, $Date$
 * @see Widget
 */
public class Button extends java.awt.Button
{
    /**
     * Constructs a Button with no label and with name "unnamed".
     */
    public Button()
    {
        this( "", "unnamed" );
    }

    /**
     * Constructs a Button with a string label and with name
"unnamed".
     * @param label the specified label
     */
    public Button( String label )
    {
        this(label, "unnamed");
    }

    /**
     * Constructs a Button with a string label and with given
name.
     * @param label the specified label
     * @param name  the specified name
     */
    public Button( String label, String name )
    {
        super(label);
        this.name = name;
        addActionListener(activateCallbackList);
    }

    /**
     * Returns a String that represents the value of this Object.
       Overrides the method in java.lang.Object.
     * @return a String
     * @see Object
     */
    public String toString()
    {
        String string = super.toString();
        int length = string.length();
        return string.substring( 0, length-1 ) + ",name=" + name
+ "]";
    }

    /**
     * Registers a callback for the top level frame. Top level
frame
       currently recognizes only ACTIVATE_CALLBACK when the par-
ticular
       instance of the button is pushed. Unknown callbacks
       are ignored.
     * @param callbackName the type of callback to register
     * @param callback     the object of the Callbackable class
that will be activated
       when triggered by the callback type given by callbackName
     * @see Callbackable
     */
    public void addCallback( String callbackName, Callbackable
callback )
    {
        if( callbackName.compareTo(ACTIVATE_CALLBACK) == 0 )
            activateCallbackList.add( callback );
    }

    // Name of instance of this class.
    private String name;

    // Callback when button is pushed.
    private CallbackList activateCallbackList = new Callback-
List();
    public static final String ACTIVATE_CALLBACK = "activateCall-
back";
}
```

Listing 6

```
import java.awt.Frame;
import com.wigitek.vivigraphics.widget.common.*;
import com.wigitek.vivigraphics.widget.gui.*;

public class ButtonTest extends Frame
{
 public ButtonTest( String title )
 {
  super(title);
```

Java for the Enterprise

New technology that will determine Java's role in the computing world

by **Ajit Sagar**

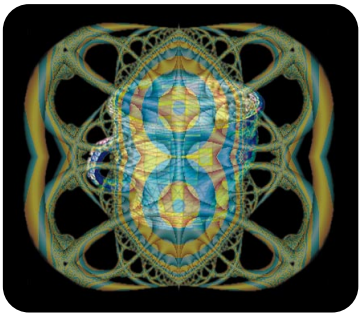
Hello and welcome to Java's Karma – The Cosmic Cup. The word Karma originates from Hinduism and means fate or destiny; or the cosmic principle according to which each person is rewarded or punished in one incarnation according to that person's deeds in a previous incarnation. In this column, we will examine Java's *Karma* – Java in its rapid incarnations; new and upcoming Java technologies that are going to determine Java's role in the computing world.

Indeed, Java has already moved beyond the realm of a programming language and holds the promise of uniting enterprise computing under one large umbrella. We will examine and explore new APIs, learn how to apply existing ones and develop example applications that illustrate how various technologies play together. I would like this column to evolve based on feedback from the readers. If you would like to see coverage on a particular topic, more information on another, or have any suggestions regarding the general theme of the column, please let me know and I will try to address your requests in future articles.

This month we will examine the role of Java in business computing, or the Enterprise. Java has, as promised, extended beyond just a programming language to a platform for distributed computing that holds the promise of solving a myriad of business problems.

When Java first made its public appearance, it was associated with a group of eleven buzzwords – *simple, object-oriented, distributed, robust, secure, architecture neutral, portable, interpreted, high performance, multithreaded* and *dynamic*. Of these, simple, object-oriented, portable, interpreted and multithreaded define the nature of the Java programming language. The Java Platform for the Enterprise attempts to make *"distributed, robust, secure, architecture neutral, and portable"* a reality. High performance is an aspect that will ultimately determine the feasibility of using Java in various applications.

The following sections discuss the APIs that attempt to fulfill the promise of a "Java Enterprise Platform." First, the Java APIs that constitute Enterprise Java are identified. This is followed by brief descriptions of these APIs and discussions of their role in the enterprise. (*Note:* I use "Java Platform" and "Enterprise APIs" interchangably. All references here to either



term should be interpreted as references to the Java Enterprise APIs.

The Java Platform for the Enterprise

So, is Java a language or a platform? Well, Java made its public debut as a language for the Internet that could be used to develop cool, interactive Web pages. But it has always been more than just another programming language. The language itself provided the core APIs in different fields of computing such as networking, graphics, etc. But it also was structured for expansion and provided hooks for plugging in enhancements and other related technologies in the fields of networking, graphics, telephony, messaging, databases, etc., which would plug and play together to solve computing problems in the business world. Many of these technologies have matured over the last couple of years and are instrumental in defining the role of Java as a platform for distributed computing.

The Java Platform for the Enterprise is a framework that will be used in developing applications for enterprise computing. In today's Internet era, that inherently implies support for a distributed architecture. The Java Platform consists of a group of Java core and extension APIs (i.e., not a part of the standard JDK) supplied by Sun Microsystems, Inc., that support this architecture. These may be categorized as:

1. **Enterprise JavaBeans™ (EJB) and related APIs:** Suite of APIs that provides extensions to existing applications and enable a separation between the business logic of a computing solution and the core infrastructure
2. **Connectivity APIs:** Suite of APIs that provides connectivity between core Java applications/applets and other frameworks in distributed computing
3. **Enterprise Services APIs:** Suite of APIs that provides computing services used to develop enterprise applications

There are other Java extension APIs that are not directly a part of the Java Platform but instead support the Platform APIs for development of enterprise applications.

The Java Enterprise APIs and their roles are illustrated in Table 1.

The Java Enterprise APIs

The following sections provide a very brief introduction to the Java Enterprise APIs. For the APIs still in their preliminary stages, an overview of the modules they support is provided.

Figure 1 shows how the Enterprise APIs form the architecture of the Java Platform for the Enterprise. The Connectivity APIs and the Enterprise Services are used to build and support the core reusable components of the Java Platform – Enterprise JavaBeans. EJBs may be used directly for the development of business applications.

Enterprise JavaBeans

EJB is the core API for the Java Platform.

It defines a server component model for building multi-tier distributed applications by combining Java components developed by different enterprise vendors. EJB component development uses all the other APIs in the Java Platform.

The EJB API has recently been released as a draft specification for public review version 0.9. Discussion of the API will be covered in a future article.

EJB architecture defines five roles in the application development and deployment workflow. These roles are described here:

- **Enterprise Beans Provider:** An applications domain expert who develops reusable EJBs, each of which implements some business logic
- **Application Assembler:** A domain expert who creates applications out of EJBs
- **Deployer:** An expert at a specific operational environment who is responsible for the deployment of EJBs and their containers
- **EJB Container Provider:** A system-level programmer who develops a scalable, secure, transaction-enabled system
- **EJB Server Provider:** A specialist in the area of distributed transaction management, distributed object and system-level services who publishes low-level interfaces to allow third parties to develop containers

Java DataBase Connectivity

JDBC is the database connectivity package now included in the core Java API. The API consists of objects used by an application to communicate with the DBMS. In the API, there are four main interfaces, based on relational database concepts:

- **DriverManager:** Enables applications/applets to download database drivers via a URL and causes establishment of a connection to the database
- **Connection:** Represents the connection to the database obtained via a getConnection() method of the DriverManager class
- **Statement:** Represents a SQL query statement passed to the database
- **ResultSet:** Represents rows of data returned from the execution of a query against the database

JDBC enables applications to connect to what is usually the nth tier in an n-tier architecture; i.e., the data source. It enables Java to be database-independent by accessing the databases via Pure Java JDBC drivers.

Remote Method Invocation

The Java RMI package is also part of the core Java API. It provides a mechanism for interaction between distributed Java objects that are similar to CORBA's IIOP. In addition, it uses Java's serialization interface to allow transfer of object instances between remote objects. RMI provides the ability for Java objects to execute methods on objects on remote Java objects and get back results from the method call. There are three main interfaces in the API:

- **Remote:** A marker (empty) interface that all remote interfaces must extend
- **UnicastRemoteObject:** Acts as a base class for server implementations in RMI
- **RemoteException:** Acts as a base class for many of the exceptions thrown by RMI remote method calls.

Using RMI involves writing a remote object and then writing its server implementation. An Object Registry runs on the server to help locate the server implementation of the remote object. RMI helps define the middle-tier in a distributed Java application.

Some extensions to RMI address reverse mapping; i.e., from the Java language to CORBA IDL.

Java Interface Definition Language

Java IDL is a Java API that allows development of multi-tier distributed applications by leveraging the standards defined by CORBA. It provides a Java mapping to CORBA and complies with the OMG standard for the CORBA 2.0 IIOP standard. This allows Java IDL programs to be used with CORBA 2.0-compliant ORBs (Object

The Java Platform APIs

Core Component APIs

API	Full Name	Role
EJB	Enterprise JavaBeans Framework	Provides access to a core set of system services for developing enterprise-level multi-tier application systems for high-volume business transactions

Connectivity APIs

API	Full Name	Role
JDBC	Java DataBase Connectivity	Provides connectivity to existing relational databases
RMI	Remote Method Invocation	Enables Java objects on one Virtual Machine to invoke methods on an object running on a remote Virtual Machine
Java IDL	Java Interface Definition Language	Defines mechanism for interoperating with CORBA for programming-language independent distributed computing

Enterprise Services

API	Full Name	Role
JNDI	Java Naming and Directory Interface	Will enable applications to access enterprise naming and directory services -- information about users, machine information and network and service
JMAPI	Java Management API	Will provide a rich set of extensible objects and methods for the development of system, network and service management solutions for heterogeneous networks
JMS	Java Message Service	Will define standard messaging services for enterprise messaging systems to interoperate
JTS	Java Transaction Service	Will provide enterprise components the ability to interface with a variety of transaction processing infrastructures

Supporting APIs

JNI	Java Native Interface	A standard programming interface for writing Java native methods and embedding JVM into native applications
Servlet API	Java Servlet API	Allows server-side Java programs to run with any major Web server and thus supports the middleware layers of the enterprise.

Table 1: The Java Enterprise APIs

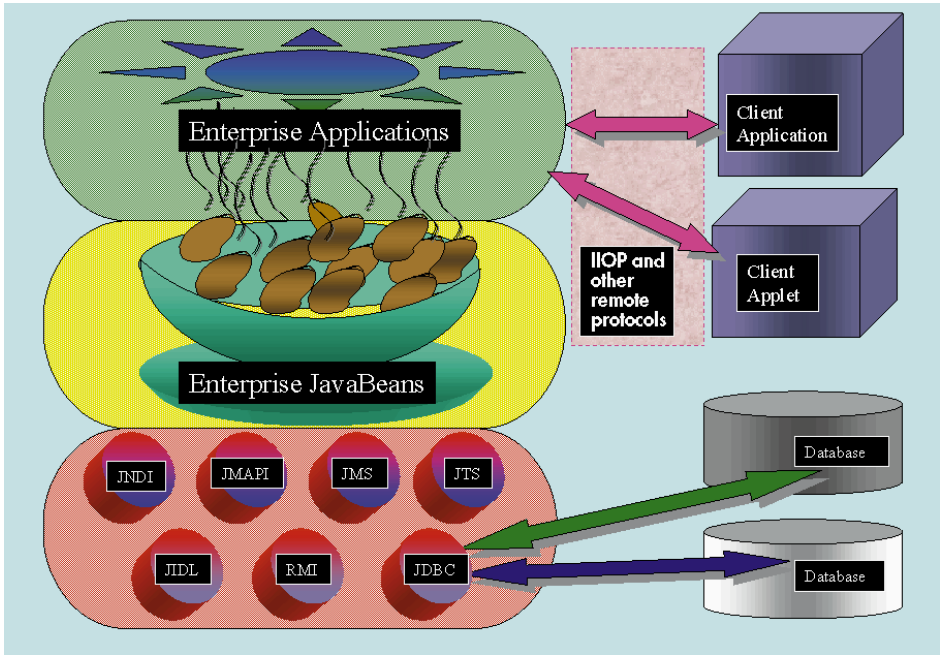


Figure 1: The Java Enterprise APIs

Request Brokers) across TCP/IP networks.

For CORBA programmers, Java provides an OMG-compliant reference implementation of an ORB. It provides the Java language mapping for the CORBA IDL.

In contrast to RMI, Java IDL enables communication between heterogeneous objects; i.e., between objects implemented in Java and other languages like C/C++, Smalltalk, etc.

Java Naming and Directory Interface

The JNDI API provides a unified interface to multiple naming and directory enterprise services. It is designed to be independent of any specific directory implementation and based on access to LDAP, NDS, NIS, DNS, etc. Standard services like name/address lookup, searches and filters are provided by the API. The main interfaces of the JNDI API are:

- **Context:** Core interface that specifies the naming context; operations such as adding/removing a name to the object-binding, looking up a name, etc.
- **DirContext:** Defines methods for examining and updating attributes associated with a directory object; also capable of providing a naming context and supporting searches
- **Service Provider Interface (SPI):** Provides the means by which different naming/directory service providers (SPs) can develop and hook up their directory implementations

The JNDI API specification is an extension API to the core JDK.

Java Management API

The JMAPI provides the user interface guidelines, Java classes and specifications for developing integrated system, network and management applications that can be used across different operating systems, architectures and network protocols. It consists of the following:

- **JMAPI Style Guide:** Provides guidelines for developing Web-based user interface in Java
- **Admin View Module (AVM):** An extension of the AWT that is specially designed for developing UI for distributed management applications
- **The Base Object Interfaces:** Supports constructing objects that represent distributed resources and services comprising the enterprise computing environment
- **Managed Container Interfaces:** Allows management applications to perform actions on a single group of managed objects, instead of each instance
- **Managed Notification Interfaces:** Provides the basic foundation from which more complex event management services can be built easily
- **Managed Data Interfaces:** Supports mapping classes and instances of the Base Objects to a relational database
- **Managed Protocol Interfaces:** Provides the infrastructure to perform distributed operations securely
- **SNMP Interfaces:** Allows extensions of the Base Objects to contain information obtained from existing SNMP agents
- **Applet Integration Interfaces:** Allow software developers to integrate their Java applets into the JavaManagement API

Java Transaction Service

The JTS is a low-level API used by sophisticated transactional application programs, resource managers, transaction processing monitors, transaction-aware communication managers and transaction managers. JTS's role is to ensure their interoperability in the Java environment. JTS is compatible with the Object Management Group's Transaction Service (OTS) specification. JTS extends the OTS specification in two areas: It incorporates the Synchronization interface defined in the in-progress draft of the upcoming revision of OTS, and it specifies how a transaction context can be propagated through other communication protocols.

JTS version 0.5 is open to public review.

Java Message Service

JMS is still in the pre-API stage. It will provide a standard for message-based enterprise communication such as:

- Publish/subscribe for smart communication between objects
- Reliable queuing
- Push/push technologies

Conclusion

In this article we examined the Java Enterprise APIs and learned the role of the Java Platform in the business industry. We briefly examined the roles played by individual APIs that make up the Java Platform for the Enterprise. Detailed information on all these APIs may be obtained from Sun's Java Website, java.sun.com/products.

So far we have browsed over the pieces that make up the Java Enterprise APIs. The larger part of this technology is still in the rudimentary stages and is continually evolving. In some of the forthcoming columns we will explore these APIs in detail and explore how these parts will come together to make the whole.

Cosmic Reflections

The Java Platform is not the first attempt in the computing industry to define a common standard for enterprise solutions. IBM tried to standardize the hardware via Personal Computers. Microsoft tried to limit the software to one platform and operating system. With the Java Platform, are we not seeing an attempt to limit enterprise computing to one "software platform?"

About the Author

Ajit Sagar is a member of the Technical Staff at i2 Technologies, Dallas, TX. He holds a B.S. in Electrical Engineering from BITS, Pilani, India and an M.S. in Computer Science from Mississippi State University. Ajit focuses on UI, networking and middleware architecture development.He has 7-1/2 years of programming experience and two in Java.

Ajit_Sagar@i2.com

1/4
Ad

The Data Series: JDBC

Offering complete database access

by **Alan Williamson**

Data! You can't live without it. Wherever you go, whatever you touch, information is continually being flashed before us. It wasn't so long ago that people were complaining of not having enough information and now our medical experts are telling us to take days off due to the information overload some are experiencing.

However, the problem isn't the volume of data that we are presented with – it's the format. We seem to spend most of our time trying to sift out the useful information from the data noise that seems to interfere with our daily lives. It is easy to blame someone else for this and not take the responsibility ourselves, while in actual fact everyone has a duty to ensure data integrity and accuracy.

This miniseries of articles looked at different ways in which data can be manipulated by the Java developer. So far, we have looked at the storage and retrieval of simple one-off data values by presenting the implementation for a simple INI file structure. Next, we looked at storing more complicated data types using Java's Object Serialization interfaces and how easy it was to store our objects for use at a later date. This month we move the series on to the next level, which explores storing significant quantities of data, enough to warrant the use of an external database. This column will cover JDBC, the Java interface for databases. We will look at what it is, how to use it and where to use it. So without further ado, let us begin.

JDBC...

JDBC, or to give it its full name, Java Database Connectivity, is a set of classes that allows easy integration between Java classes and external databases. JDBC presents the user with a consistent interface to a database, irrespective of the data engine. For example, JDBC affords the developer the ability to test and develop using an Access database, and for their application to ship which then talks to an Oracle database, all without having to recompile a single line of code. It is this power that has made JDBC one of the most used APIs of the complete JDK package list, and once a developer has mastered the few classes that make up the JDBC package, there is no looking back. So how does it work?

JDBC can be broken into two distinct parts: the developers API and the service providers API. The developers API is what we are presented with in the JDK, and it is this one we use to insert and query data into our tables. The service providers API is the other side of the equation. It is this API that a service provider implements to develop a JDBC driver which sits in between JDBC and the actual database. For example, Oracle provides a free JDBC driver for use with their database, while Sybase provides one for use with theirs. When a developer wishes to change the database engine they are using, they merely replace the JDBC driver. The majority of mainstream databases have JDBC drivers available for them, including the standard ODBC driver that ships with the JDK.

JDBC uses the standard query language, or SQL, to facilitate the communication between the developer and the database, performing all translations transparently. For example, Microsoft uses a different flavor of SQL from Oracle. Some queries simply will not work in Access but will operate perfectly in Oracle. Fortunately, Oracle implements the true SQL interface, but as a developer you don't need to worry about these inconsistencies. Assuming your SQL is standard, JDBC will perform any conversion that may be necessary between different database standards.

We will look at the major stages in developing communications with a database: connection, querying and updating.

Connecting

Before you start using any tables, you must first make a connection to the driver, which in turn makes a connection to the actual database. The best way to illustrate this procedure is to use an example. The code below shows the basic stages in setting this up:

```
Class.forName( "sun.jdbc.odbc.JdbcOdbcDriver" );
Connection con = DriverManager.getConnection(
"jdbc:odbc:testdb", "nary", "ceri" );
```

The first call ensures that the class loader can find the JDBC driver that we intend to use. In this instance we will use the standard one that ships with the NT/95 version of the JDK, which uses the JDBC-ODBC bridge for the connection manager. Having successfully found the correct class, the *DriverManager* class attempts to retrieve a *Connection* instance to the specified database. This can be thought of as a login session to a given database. It is not uncommon for several connections, or sessions, to be open at once. Notice the parameters to the *get-*

Listing 1.

```
Statement Statmt;
ResultSet Res;

Statmt = Con.createStatement();
Res    = Statmt.executeQuery( "SELECT FORENAME,SURNAME,AGE FROM
CONTACT WHERE FORENAME LIKE 'A%'" );

while (Res.next()){
  System.out.println( "Forename: " + Res.getString(1) );
  System.out.println( "Surname : " + Res.getString(2) );
  System.out.println( "Age     : " + Res.getInt(3) );
}

Statmt.close();
Res.close();
```

Listing 2.

```
Statement Statmt;
ResultSet Res;

Statmt = Con.createStatement();
Statmt.executeUpdate( "INSERT INTO CONTACT(FORENAME,SURNAME,AGE)
VALUES('Alan','Williamson',78)" );
Statmt.close();
```

Connection method; we have a URL to the database, a user name and password. One of the advantages of addressing the database in this manner is that it allows for the database to reside on a remote machine. Having retrieved the *Connection,* we can now start to query and use the database.

Querying data

One of the most common functions a database performs is that of holding data for possible future use; the way in which we retrieve data is through the use of SQL Select statements. We create the SQL statement using standard ASCII strings and then pass them to JDBC, which passes them on to the database for execution.

SQL presents the results from such queries as a table which can be read column by column, row by row. JDBC provides a class, *ResultSet,* to make reading this result table a trivial matter. Listing 1 is an example of reading all the results when the table is queried for all names beginning with 'A'.

The *ResultSet* is a class that maintains a cursor into the results table, which can be advanced using a call to the *next()* method. An important thing to remember is that once a row has been advanced, it can not be revisited. The query would have to be rerun in order to read the row again. Retrieving individual columns can be achieved using a variety of *getXXX(…)* type methods from the ResultSet class. For instance, in our example we use both the *getString(…)* and *getInt(…)* class to retrieve all the column values. The columns are retrieved using column indexes, starting at 1, and occurring in the same order they are specified in the SQL statement. It is important to use the correct *getXXX(…)* method for the column data, since a some castings will throw an exception. For example, if you try to use a *getLong()* method to retrieve a column with text in it, then an exception will most definitely occur. However, you could use a *getString(…)* method to retrieve a column that has nothing but integers in it. Just be careful when choosing your retrieval method.

Storing/Updating/Deleting data

Storing data is as straightforward as retrieving it. Using the SQL statement INSERT, we can pass data for inclusion via JDBC. We can build up the SQL INSERT statement, using standard ASCII strings, as in the example shown in Listing 2.

Once the statement has been constructed, it is run using the *executeUpdate(…)* method. This will return a variety of different results depending on the SQL being run. In this example, the method would return the row number where the insert took place.

Using these same procedures, we can use the *executeUpdate(…)* method for both the other popular SQL statements: UPDATE and DELETE.

Summary

This article took a whistle stop tour of the functionality provided by JDBC. The main reason for this was to introduce JDBC to you before we look at the final article in this miniseries, the database layer provided by Symantec's Visual Café. This mini-series explained the whole issue of data storage and the differing options depending on the volume of data being handled. As each article progressed, the level of complexity also increased, culminating to complete database access, using JDBC.

In the next column we will look closely at how Symantec has provided a complete set of wrapper classes that makes even the simple interface of JDBC even simpler to use.

About the Author

Alan Williamson is on the Board of Directors at N-ARY Limited, a UK-based Java software company, specialising solely in JDBC and Java Servlets. He has recently completed his second book, focusing purely on Java Servlets. Alan can be reached at alan@n-ary.com (http://www.n-ary.com) and he welcomes all suggestions and comments.

alan@n-ary.com

1/2 Ad

Singletons, N-tons & Static-only Classes

The whys and hows of making finite-instance classes

by **Brian Maso**

Often it's very useful to create classes that represent a restricted number of real-world entities in our Java programs. For example, there is only a single instance of the java.awt.Toolkit class that ever exists in a single Java VM. This Toolkit represents the windowing system on the local machine. It would generally not make sense to have more than one Toolkit object, so the Toolkit class is designed to ensure that only one instance can ever be created. The Toolkit class is an example of what is termed a Singleton class.

Java classes have the interesting capability to represent singleton resources themselves. That is, you could use a static-only class to represent a single resource. The best example that comes to mind in Java's Core API is the System class. The System class represents the O/S information and functionality available to your Java application: system properties, loading dynamically loadable libraries, access to the VM's garbage collector, etc.

Singletons are a specific example of a more general concept: the N-ton class. The N-ton class is a class in which only N objects are created; N being a finite number. (A singleton class is an N-ton class where N equals one.) An example of an N-ton class would be a class to represent the serial ports available on a given machine. On a given machine there are a finite number of serial ports which you can use to communicate with serial devices, like modems, Iomage Zip drives, etc. It is easy to imagine a Java class that represents the serial ports available. You would want to design this class so that only a finite number of objects of that type would be created.

In this month's column I'm going to discuss the features of N-ton classes verses static-only classes and talk about when it is appropriate to use either approach.

Starting with Static-Only Classes

The primary feature of a static-only class is that, like the term implies, all of the members of the class are declared as "static". There are no instance variables or instance members for the class. Instead, the state of the resource being represented is stored in static variables of the static-only class. The functionality of the resource is exposed through the static methods of the static-only class. Using the example of the java.lang.System class, the system's "state" is represented using the static variables of the System class. The variables "in", "out" and "err", for example, are used to access the standard input, output and error streams of the current process, respectively. All three of these variables are declared as static.

The point of a static-only class is that you never actually create an object of that type. Instead you use the static members of the class very much like you would a singleton object. It is easy to ensure that an object of a static-only class is never created. You have two options available to do this (or you could use both, although that's overkill):

1. Declare the class constructor as "private". This ensures that no code external to your class can create any objects of that type; the compiler simply won't let you. However, this does not restrict code within the static-only class from creating a new instance. That's why I prefer the second option.
2. Declare the static-only class as "abstract". No code, even code within the static-only class itself, can create an object of an abstract type. So, adding the "abstract" attribute to your class guarantees no instance will ever be created.

Static-only classes are not used in other popular object-oriented languages mainly for one reason: How do you initialize static variables? That is, static variables are, by default, initialized to zero-states (numbers and chars are zero, booleans are false and object references are null). In languages like C++ there's no easy way to initialize the static variables of a class without writing and calling special methods to do so.

The Java language provides us with the "static initializer block" which allows you to initialize static variables in a class when the class is loaded into the VM. So, if I was writing Java's java.lang.System class, I would add a static initializer block in the class to set the "in", "out" and "err" static variables to non-null object references. When the System class is loaded, the static initializer is automatically run, which would guarantee my static variables would never be null. Listing 1 shows an excerpt of the System class I would write. The static initializer block is the block of code preceded solely by the keyword "static".

N-ton Classes

N-ton classes, including singleton classes, are a bit different than the static-only classes. The idea of an N-ton class is that you do create instances of the class, but you want to guarantee only a finite number are ever made. So the features of an N-ton class are a little different than static-only classes. N-ton classes usually have private constructors (or, in some cases, package-private or protected, but for simplicity I'll stick to "private"). You get a reference to one of the N class instances through one of two different mechanisms. The first is a static "lookup" method; a method that takes some sort of identifier (a String or ID number, for example) as a parameter and returns an instance for that identifier. This is the way to do it if you don't necessarily know the number of objects at compile time. For example, the serial port class I mentioned earlier might have a lookup method with this signature:

```
// static Lookup method for a SerialPort
class
public static getPort(int portNumber) {...}
```

The second mechanism for obtaining a reference to an N-ton object is through a set of N static object reference variables of the class type. This would be the way to do it if you know the number of objects at compile time. These N static variables could be initialized using a static initializer block the same way a static-only class would use to initialize its static variables. Listing 2 shows example code for this kind of N-ton class.

An example in the Java Core API of an N-ton class is the java.net.InetAddress class. For this class, N is huge (the number of host

machines on the Internet, which numbers in the millions), but it is finite. The InetAddress class has both features I've listed for N-ton classes: a restricted constructor and a public static "lookup" method.

An N-ton setup is clearly the way to go if N is greater than one. Which begs the question: "Should I use a static-only class or an N-ton class when N is equal to one (i.e., for a singleton class)?". The answer, just as clearly, is: "It depends." Either technique would work, in theory. But one attribute makes it easy to differentiate which technique you should use for singleton classes. That attribute I call "replaceability".

Replaceability is the ability to replace a singleton's implementation at runtime. The java.awt.Toolkit class is a great example of a singleton with a replaceable implementation. The Toolkit class itself is an abstract class. The idea is that different VM implementors will provide a different (that is, replacement) implementation for the Toolkit on different systems. Microsoft provides a Toolkit-derived class to be used with their VMs on MS systems, and Sun provides their own Toolkit-derived class to be used in their VMs. The Toolkit implementation needs to be replaced with a particular implementation on different systems.

In cases where replaceability is required, it is most appropriate to use the N-ton class design and less appropriate to use the static-only design. The Toolkit class itself has both of the features I listed above for N-ton classes: a restricted constructor and a static lookup method (called Toolkit.getDefaultToolkit). In Listing 3 I show my implementation of the getDefaultToolkit method of the Toolkit class. This method finds the correct implementation of Toolkit functionality to use on the current system by looking for the system property "java.awt.toolkit" and using its value as the name of the replaceable implementation.

The System class, on the other hand, is a static-only class to represent a singleton resource. In this case, replaceablity is not a concern so a static-only implementation is used. The System class itself has the features listed above for static-only classes; a private constructor and the class public interface includes only static members and methods.

There's one other major difference between the static-only design and the N-ton design. That difference is that N-ton object instances are finalized, which allows you to clean up allocated resources. Static-only classes, on the other hand, generally never experience finalization and so have no opportunity for class clean up. In next month's column I'll discuss object and class finalization, which should help explain this key difference more clearly. I'll also show how to approximate static "uninitializer" blocks in your Java classes.

About the Author

Brian Maso is a Java programming consultant and president of Blumenfeld & Maso, Inc. He works out of Dana Point, CA. He is the author of several Java books. Before Java, he spent five years corralled in the MS Windows branch of programming, working for such notables as the Hearst Corp., First DataBank, and Intel. Readers are encouraged to contact Brian via e-mail with any comments or questions at bmaso@developer.com.

bmaso@developer.com

Listing 1: My implementation of part of the java.lang.System class.

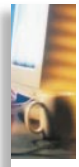
```
//Note use of static member variables, a private constructor and a
//static initializer block.
//NOTE: This not how the Java Core API source code implements
//this class. This is to demonstrate using a public static
//interface to make a static-only class.

package java.lang;

import java.io.*;

public class System {
  public final static InputStream in = getIn();
  public final static PrintStream out = getOut();
  public final static PrintStream err = getErr();

  static {
    // Load native implementation of getXYZ() methods
    loadLibrary("System");
  }

  private static native InputStream getIn();
  private static native InputStream getOut();
  private static native InputStream getErr();

  private System() { }

  ...
}
```

Listing 2: Implementation of an N-ton class.

```
//Features include restricted constructor, finite number of public
//static objects representing the N objects, static initializer
//block initializing the objects.
public class Color {
  int red, green, blue;

  private Color(int r, int g, int b) {
    red = r;
    green = g;
    blue = b;
  }

  public static Color Red;
  public static Color Green;
  public static Color Blue;
  public static Color Purple;
  public static Color Orange;
  ...

  static {
    Red = new Color(255, 0, 0);
    Green = new Color(0, 255, 0);
    Blue = new Color(0, 0, 255);
    ...
  }
}
```

Listing 3: Part of the Toolkit class. Uses N-ton pattern to support replaceablity.

```
package java.awt;

public abstract class Toolkit {
  ...

  private static Toolkit theToolkit;

  static {
    try {
      String toolkitClassName =
          System.getProperty("java.awt.toolkit");
      Class clsToolkit = Class.forName(toolkitClassName);
      theToolkit = clsToolkit.newInstance();
    } catch (Exception e) {
      theToolkit = null;
    }
  }

  public static Toolkit getDefaultToolkit() {
    if(theToolkit == null)
      throw new RuntimeException("No toolkit exists!");

    return theToolkit;
  }

  ...
```



Visual Café for Java Database Development by Symantec

An Integrated Development Tool for Creating Java Applets and Applications that Interface with Databases

by **Dana Crenshaw**

Visual Café for Java
Symantec
10201 Torre Avenue
Cupertino, CA 95014
Phone: 888 822-3409
Fax: 408-253-3968
Web: www.symantec.com
Price: $799.95

Visual Café for Java is an integrated development environment tool for creating Java applets and applications that interface with databases. In addition, you can create JavaBeans, native EXEs and DLLs. Some of the improvements include:

- Contextual menus
- ZIP archive and Java Archive support
- Java version compatibility with JDK 1.0.2
- Customizable Project windows
- New and improved wizards
- Incremental debugging

Test Environment

486DX4/100, 24MB RAM, 1.03 Gigabyte Harddrive, Windows 95, IBM SVGA monitor and 4X CD-ROM.

System Requirements

Minimum system requirements for use of this product are:

- Windows (95, NT Workstation version 4.0+ or NT Server version 4.0+)
- Pentium Compatible
- 24Mb of RAM for Win 95 (32 Mb RAM recommended)
- CD-ROM
- Color Monitor with 256 colors capability
- At least 60 MB of harddisk space

Required harddrive space depends on which tools you install and whether you have FAT16 or FAT32.

I strongly recommend installing on a Pentium computer with at least 32Mb of RAM. I installed on a 486-based PC because that is what I have. If you only have a 486, I would suggest you also have at minimum 32 Megs of RAM; more would certainly be better. The speed was slow but tolerable. Symantec recommends a Pentium with a CPU speed of 90 Mhz or higher. Take their advice.

Product Installation

Installation of Visual Café follows your typical Win 95 software installation process. The customized installation took approximately 20 minutes. The only options I excluded were the sample applications and the Tour software. A full installation requires 156 Megabytes (86 meg for program files), 13 meg for help, 22 meg for JFC (Java Foundation Class) and 22 meg for samples files (I'm not sure where the remaining 13 bytes went).

Figure 1 shows you the software that you have a choice of installing. At minimum you need to install Visual Café 2.0 and dbANYWHERE Server. The remaining packages are for creating and publishing HTML (Visual Page), browsing the Internet (Netscape Communicator), Server software (FastTrack) and SQL database (Sybase SQL Anywhere).

This version of Visual Café supports JDK (Java Development Kit) version 1.1. As you

Figure 1. Installation Screen

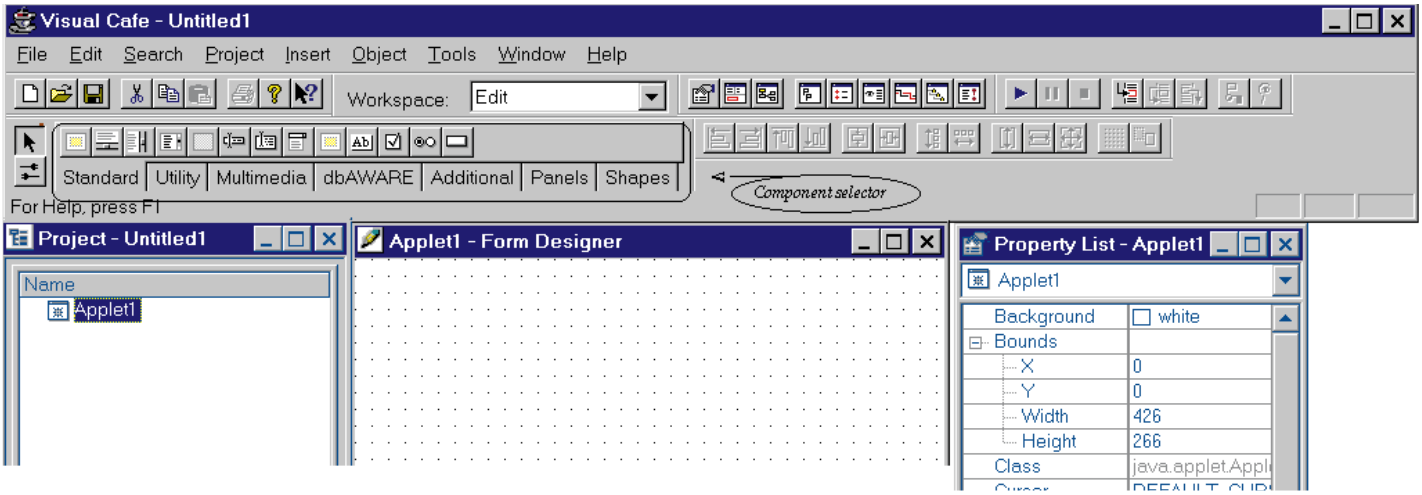


Figure 2. Visual Cafe Project Window

can see, there is a patch for the Netscape Communicator in reference to the JDK. It was not made clear whether or not the JDK must be already installed. This is not something to take for granted because I had previously installed version 1.1.3 to create applets. Just to be safe, I zapped JDK 1.1.3 from my hard-drive. Visual Café does come with all of the necessary Java software, and so I did not run into any compatibility problems.

About Visual Café 2.0

Visual Café DDE (Database Development Edition) is the must-have tool for developing Java applets and/or applications with database connectivity. It comes with several wizards and tools that make development seem almost effortless There is the dbNAVIGATOR window which enables you to access metadata through the dbANYWHERE server. In this window you can view databases in your server and drill down to look at data in the database tables. Also, from this window, you can drop database objects right onto your form. Visual Café will automatically plug in properties for the components with information from the database.

Next, there is the dbAWARE Project wizard which helps you to set up your project. Utilizing a series of screens, the wizard helps you to define your data source, choose which tables you are going to use and what columns from those tables you will select data from. Upon completion, the created form is displayed in the forms editor.

Figure 2 represents a new project. On this screen you have four basic areas: the toolbar, project component list, applet form and component properties box. I do not have enough space to justly define their full capabilities here – you really have to try this product for yourself – but, nevertheless, I will go over them briefly. The component list displays a listing of components (labels, textfields, panels, etc.) being used in the applet. The applet form is where you place the components for the applet and the properties box allows you to set the properties for each component. This properties box is a gem.

If you have ever developed Java applets/application without the use of an IDE, you know it can be very tedious. You will love the properties box. By simply clicking on dropdown menus or typing in numbers, you can easily set properties. For example, to set the layout manager to GridLayout, you select it from a dropdown and then set the number of rows and columns in the proper fields. What about setting the layout to GridbagLayout? You'll have no more nightmares of messing around with the constraint object. It's a breeze with Visual Café.

I saved the toolbar for last because it is so special. I am calling the circled area 'Component Selector.' It consists of tabs that dictate which components you have available for placing on your form. You have Standard, Utility, Multimedia, dbAWARE, Additional, Panels and Shapes. Each tab yields a different set of components, represented by the icons placed above the tabs. If you are not sure what the icon represents, you can place your cursor over the particular icon and help will pop up, displaying its identity.

Documentation

Accompanying the CD-ROM containing the software, are three manuals. The first is a user guide for Visual Page. The remaining two are titled "Getting Started" and "User's Guide". These two are for Visual Café. "Getting Started" tells you about installation and how to take the tour. The tour introduces you to Visual Café. If you are new to Java development, I recommend taking the tour.

The "User's Guide" contains a lot of useful information. Of course it covers all of the functionality of Visual Café. However, it is not a manual on the Java language and was not meant to be. To use Visual Café you do not have to be a Java expert, but you should have some familiarity with Java.

You can use Visual Café as a learning tool. If you have some experience with Java but not a lot, you can study the created ".Java" file to learn how applets are created. You will see how to set properties for components, how to use Panels and even how to write code that handles errors. You will also see how to create Java classes for applets and applications. Visual Café gives you the ability to modify the code.

Recommendation

Simply put, if you are serious about developing Java applets and/or applications then you must avail yourself to Symantec's Visual Café. I have had brief experiences with Microsoft' Visual J++ and Powersoft's PowerJ, and I rate Visual Café as the best by far. When it came to creating applets I had given up on IDEs because they appear clumsy and stifling. However, that has changed with Visual Café. If you try it, you will get hooked just as I am.

Editor's Note:

Visual Café for Java 2.5 is shipping in late March. The 2.5 features include:

- JDK 1.1.5
- Universal interface with MDI and SDI support
- JavaBeans Wizard
- Full dbANYWHERE server with unlimited connections
- RAD on/off
- Symantec's JIT (Just In Time) Compiler 3.0.

About the Author

Dana Crenshaw is a freelance writer based in Atlanta, Georgia. He is a systems analyst with over 13 years of experience. Dana can be reached at DanaP@CompuServe.com.

DanaP@CompuServe.com

OrbixWeb 3.0

by Iona Technologies, Inc.

A 100% Pure Java ORB with Better Lifecycle Management of Distributed Objects

by **Khanderao Kand**

OrbixWeb 3.0
Iona Technologies, Inc.
60 Aberdeen Avenue
Cambridge, MA 02138 USA
Phone: 617 949-9000
Web: http://www.iona.com
Email: info@iona.com
Price: $799 Standard Edition, $1,499 Pro Edition
Platforms: More than 20 platforms uncluding Solaris 2.x, HP-UX, AIX, IRIX etc. Windows95 & NT

OrbixWeb 3.0 is a CORBA 2.0 compliant, 100% Pure Java ORB with better lifecycle management of distributed objects. It includes many useful services which make it easier to develop, deploy and manage distributed applications in the Internet age.

In today's computing age of Internet and multi-tier architecture, combining Java and CORBA offers the best solution. If you want to do it, then you might want to look into Iona's OrbixWeb 3.0 as one of the leading edge, robust and proven solutions. OrbixWeb is a fully CORBA 2.0 compliant distributed application development environment for Java.

Iona and CORBA

In the CORBA world, Iona has many firsts to its credit – including the first commercially available CORBA compliant ORB in 1993. With a strong research background of founders, it has always been a technology leader. In OrbixWeb version 3.0, Iona has removed remaining C++ code from its support utilities. The basic ORB was already Pure Java. This version comes with many new services, improved server object invocation and OMG-compliant Java mapping.

Basics of OrbixWeb Architecture

Orbix ORB is implemented as a pair of libraries (for client and server) and the Java based activation daemon orbixdj (orbixd for C++). The daemon is needed only on servers' hosts. It is primarily responsible for (re)launching server processes and connects the client/server for the first time. The daemon maintains a simple database called Interface Repository which maintains the activation modes, access lists and other information about the server. In case of "in process" activation, no separate process is started but the server runs within the same VM (virtual machine). This improves performance. For non-Java servers you will have to use "out process" activation. Manual activation would be required for persistent servers and you might use it for your legacy server. Orbix supports different thread models – namely, per-client, thread-pool and per-client-thread.

After establishing the connection, the requests are passed directly from the client to the server. The messaging component within the libraries manages the (un)marshalling, protocol and synchronization. Orbix supports its proprietary and OMG IIOP protocol.

OrbixWeb-Based Distributed Systems

As a CORBA-based developer you might need to begin with writing the interfaces, using OMG's Interface Definition language, for your object.

Then you compile the IDLs using Iona's supplied IDL-to-language compiler. The IDL to language compiler (Java, C++, Smalltalk, Ada) generates (client side proxy) stubs and (server side) skeletons in that language. OrbixWeb 3.0 completely supports the OMG IDL to Java mapping including the Java ORB portability interfaces. You don't have to recompile stub and skeleton for use against different vendor's ORBs. If you use a previous version of OrbixWeb then I have bad news for you. The new Java IDL mapping is different from OrbixWeb's previous version. You must convert to the new interfaces. The good news is that you can use migration utilities.

Develop Dynamic Client and Servers

In a different scenario, you might not need to start with writing IDL; that is, using Dynamic Invocation Interfaces (DII). In this case, your client might not know the interfaces at compile time. In such cases, the client program browses the Interface Repository (IFR), gets the methods and parameters, formulates the request, sends the request and then polls and processes the result. You can use trader service (OrbixTrader) and naming service (OrbixNames) for getting the object reference that is required for IFR. OrbixWeb also supports Dynamic Skeleton Interface (DSI) which is for developing a dynamic server.

Develop Web Applications Using OrbixWeb

OrbixWeb 3.0 supports Internet Inter-ORB Protocol (IIOP) communication. It enables the Object invocation and callbacks to be sent across the Internet. A client applet along with any browser can download the stub and related OrbixWeb classes. OrbixWeb does not depend upon static embedded browser classes or plug-ins. Thus, if required, you can change the ORB configuration information on-the-fly. The applet then uses Internet ORB Protocol (IIOP) to communicate with the backend services on the Web server.

One of the most serious concerns you might have for Internet applications is security. To protect the internal hosts from the hacker's attack, you might need to use Iona's Wonderwall. Wonderwall is a mechanism for firewall navigation by IIOP requests. This proxy allows IIOP traffic to be filtered and routed at the firewall. The Wonderwall "proxifies" the Interoperable Object reference (IOR) of an object and

AD

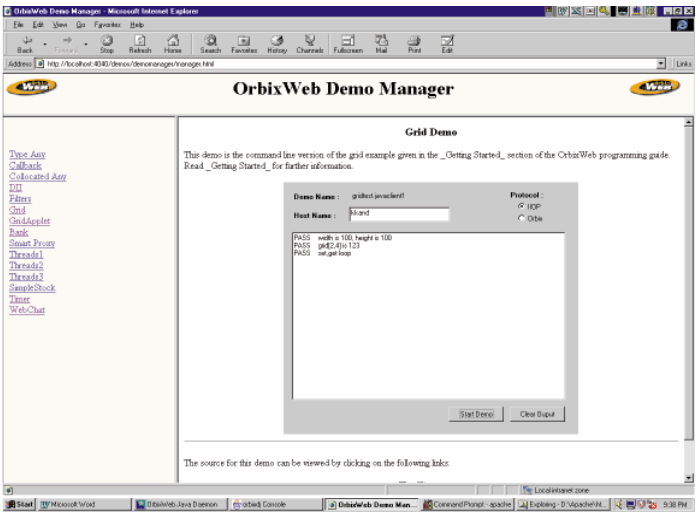


Figure 1: Using OrbixWeb for Web

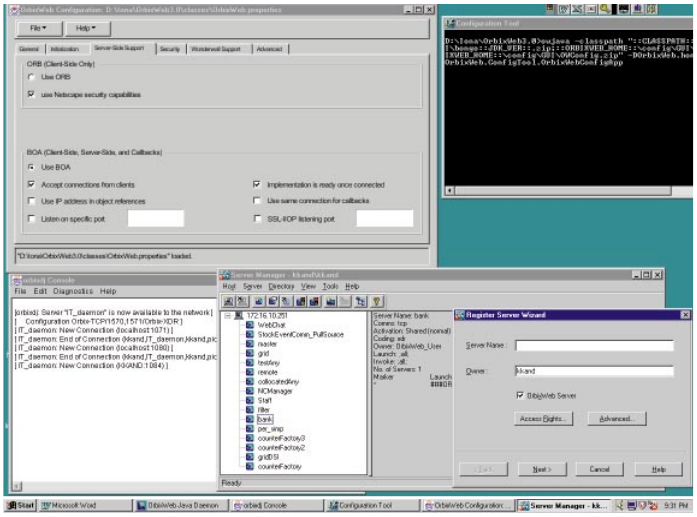


Figure 1.2 Configuration Tool and Server Manager

allows remote invocation on the proxy IOR. Once a proxy for the IOR has been established, the client can communicate with the proxy through a single, known port.

If your client is beyond another corporate firewall, OrbixWeb tunnels your IIOP request through HTTP. Basically your client's request gets "wrapped up" in HTTP by the applet's ORB classes. This is known as "HTTP tunneling".

In addition to the wonderwall, you can use OrbixSSL to gain more security. OrbixSSL API (available separately) uses 128bit encryption with DEC, RC4 algorithms. You can use OrbixSSL to replace your default IIOP with the SSL-IIOP protocol.

Iona's Strong Base of Services

Orbix implements many important CORBA compliant services. OrbixEvent implements Event specification. OrbixTalk is a messaging support system based on IP multicast. For IBM customers, Orbix offers Orbix+MQSeries, bringing the power of IBM's messaging middleware in a CORBA environment. OrbixTrader, a CORBA-compliant trader service, is like the yellow pages. OrbixNames is a naming service, which binds meaningful names to object references. For database oriented systems, Orbix comes with OrbixOTS, which implements CORBA transaction services. Orbix has excellent adapters for object databases like Versant, ObjectStore and other ODBMS.

Customize the ORB with Application Integrators

OrbixWeb provides a wide range of integration hooks which give you control and flexibility over the ORB:

- Loaders – Implement your persistence mechanism using loaders.
- Filters – Expose the request path before, and after marshaling. Implement debugging authentication mechanisms.
- ThreadFilters – Use it to configure a server side thread model to thread-per-request, thread-pool or thread-per-client.
- Transformers – Used for facilitating data privacy
- Callbacks – Use client callback interfaces which a server can invoke without your client explicitly requesting information
- Locators – Override the mechanism for determining target object locations using locators API
- Smart proxies – Override the client stubs. Typically, you can use it to improve performance by implementing client-side caching. You can otherwise use smart proxies for sever rebinding (fail safe), breakpoints or for non-supported type conversions (legacy systems).

Installing, Configuring and Managing OrbixWeb

The installation of OrbixWeb is straightforward. I installed it on my NT boxes. The configuration was easy too. The GUI-based, user-friendly configuration tool would make your administration easier. You can change the properties of ORB and wonderwall on-the-fly. OrbixWeb maintains the properties as properties class.

You can also monitor your server objects through a GUI-based server manager. Orbix provides different access privileges for clients to invoke, launch or use the server. You can also register the server as one of different invocation modes, shared modes or unshared (per client) or per method (invocation). The secondary modes could be multiple client, per client or per client process. You can maintain server privileges like owner, launch or invoke privilege. All of this gives you better control over server object management.

My Concerns?

- Licensing: Iona charges $100 per runtime for multi-processor machines. Single-processor runtimes are free.
- Firewall: The Wonderwall approach was tightly integrated with the Orbix.
- Debugging: Using APIs we might develop and integrate debugging support, but it would have been nice to have seen it built-in with the product. I hope to see RDBMS adapters very soon.

Orbix is one of the few to support both client-side and server-side ActiveX, COM components. It enables you to develop Visual Basic, PowerBuilder and ActiveX as both client and server objects.

What Did I Miss?

1. Performance: You might need to do some testing of performance reliability before selecting any ORB. I found OrbixWeb to be pretty good.
2. Comparing vendors: The comparison of different ORB vendors can be scaled based on different factors. Such a comparison requires a separate effort.

OrbixWeb: A Flexible, Easy-to-Use and Robust CORBA Framework

For a long time, Iona has maintained a technological edge in the CORBA market. This experience has enriched Iona's expertise in many industries. Orbix offers better control over the ORB and server management. It also provides more services than other vendors. Allowing applets as both client and server objects, Web naming, Wonderwall and SSL, Orbix is suitable for Internet. Recently, Iona has licensed COM from Microsoft. We can definitely expect the best integration of both the technologies from Iona in future.

About the Author

Khanderao Kand is presently working as Senior Software Engineer at Arbor Software. He can be contacted at kkand@arborsoft.com

kkand@arborsoft.com

Tips for Developing Pure Java Applications

An emphasis on delivering pure Java code offers substantial benefits

by **Bob Adams**

When I recently embarked on Java development for some telecommunications clients, I quickly learned that pure Java code is not a PR fad, it is a technical necessity. A few short months after Sun announced the 100% Pure Java initiative, I was grappling with development tools and third-party code that was itself in transition. Here are some tips and comments based upon my real-world experience at producing and verifying pure Java code.

The rationale for committing to Java in the first place was very clear. I worked for a firm that was growing nearly 100 percent each year as it developed both a professional services organization (PSO) and a practice specializing in call centers solutions (CSC). The PSO, among other things, targeted building Web-enabled client/server applications that linked databases to the Internet and corporate intranet.

I was refining the client software for use by telecommunications firms, as well as the middleware infrastructure for tying together legacy applications with distributed desktops. Telecommunications is a highly competitive market where customer care and customer service are increasingly important criteria for differentiating one firm from another. I was providing client software with four foundation modules – subscriber, agent, sales management and administration – plus many options for customization and custom add-ons. Telecom firms typically have a lot of downstream legacy systems and databases that have to be integrated into these client modules, so engineers were working with Tuxedo transaction monitors, Oracle databases and CORBA-based messaging.

In such an environment, I had to deliver a very thin client application that was platform-independent. Users would be distributed throughout both corporate intranets and to subscribers on the Internet. There was simply no way to know whether users had PCs, Macs, workstations or network computers. Pure Java code was a technical necessity.

That said, the 100% Pure Java initiative was only a few months old. Some tool vendors and many suppliers of third-party modules were still getting up to speed on ensuring the code they generated or delivered was indeed Pure Java. Engineers worked through code glitches and, in some cases, assisted third-party partners in making sure their code was fully compliant.

Benefits

Corporate developers and development partners should not pigeon-hole the Pure Java initiative as merely an independent branding program for software vendors. Java Pure Check, which is freely available in conjunction with the initiative, is an essential tool for in-house testing. And as the certification program continues to evolve, it is becoming far easier for internal development teams to insist that third-party code modules they bring into their firm are truly 100% Pure Java code.

"With the proper foundation, an emphasis on delivering Pure Java code wherever possible can offer substantial benefits to the developer."

Even if there is no intention to verify the purity of Java code with an independent testing lab, using Java Pure Check should be part of any good quality assurance process. Testing all classes and methods is simply an excellent requirement, especially as hectic development schedules stretch resources and create a temptation to look for short cuts, or as development teams expand. My experience confirms that the Pure Check process provides an excellent quality assurance building block that is very consistent with ISO certification requirements that methods be in place to guarantee all software is tested before deployment.

It is also important to recognize that the inherent capabilities of Java, JavaBeans and Enterprise JavaBeans make possible a fundamentally superior testing methodology for corporate developers and large integrators. Developers can not only separate server and client applications more distinctly than ever before, they can also separate out business functionality from enterprise plumbing issues on the server side.

Testing procedures can, therefore, focus not so much on testing applications as they do on testing components. Shifting the focus to component testing enables a team to verify functional chunks and should dramatically boost productivity.

Clearly separating application components also makes it far easier for development teams to roll out new functionality. Complex enterprise applications will include Java and non-Java elements. Java-based functionality can be added incrementally over time, as well as upgraded, over time. Adhering to well-defined interfaces is crucial to leveraging that flexibility.

For my clients, there were also strong business drivers behind the commitment to Java in general and to the specific commit-

ment to ensuring the code met Pure Java standards. Java is platform-independent, it is secure, it is ubiquitous and it solves the client software distribution problem for network-centric applications. Ensuring that Java is always Java – that it is 100% Pure – is essential to delivering the goods. It is also these inherent qualities that make possible thin-clients, including network computers.

Whether on a large intranet or an extranet where business partners and customers access a firm's computing resources, companies want to minimize release management headaches. From the users' perspective, whether the application works properly will shape the perception of the company's reliability, so regardless of the client platform, the software needs to run well. For telecommunication firms that means there are some pretty expensive ramifications to inconveniencing customers or business partners that go well beyond the well-documented impact of superior customer service on customer loyalty.

Pure Java Requirements

Java Pure Check focuses on the purity of Java code rather than the portability of program functions. Purity is a far more objective measure and Pure Java programs substantially reduce the risk of portability problems now and in the future.

Sun defines a Pure Java program as one that relies only on the documented and specified Java platform. It is, therefore, a self-contained set of classes with no external dependencies other than the Java Core API. A Java program may be an application, an applet, a class library, a servlet, a JavaBeans component or more than one of the above. For formal certification, if an application uses libraries outside the Java Core API, those libraries need to be packaged with the application and must be 100% Pure Java as well.

Purity is measured at the bottom edge of the program where it interfaces with the platform, rather than at the top edge where it interfaces with the user. As such, purity is a good predictor of portability and a pure program should not be accidentally unportable. Programs can be checked for purity individually, so client and server side Java programs can be evaluated in a completely independent manner.

As the above information indicates, there are some hard and fast rules about writing Pure Java applications. The first is that there can be no native methods in the application.

Introducing native code into a Java program sacrifices most of the benefits people look to Java to deliver – security, platform independence, garbage collection and easy class loading over the network. The security implications for users can be very significant, wiping out assurances that the code is free of viruses and making it far more likely that memory problems – pointer overrun or attempting to access protected memory – can crash the Java Virtual Machine.

That said, native method definition and some methods in the java.lang.Runtime class do provide access to hardware-specific code, which is useful for Java programs that access legacy systems. By definition, however, the interface programs cannot be Pure Java.

Where it seems necessary to include native methods to deliver access to a system resource that is not properly supported by Java, or to boost performance, there are a couple of alternatives. One is to define a simple protocol to give that service and then write a Pure Java program as a client of that protocol. Another is to rewrite the native method in Java.

The Java Native Method Interface (JNI) does not make native code platform-independent, although it does make it easier to port native code. The native code still must be recompiled for each different hardware

Bristol
1/2 Ad

and that recompilation will be difficult or impossible if the target hardware does not provide the library or the capabilities required by the native method.

A second rule is that applications must depend only on the Java Core APIs and should not depend upon the internals of any particular Java implementation. Standard classes and standard interfaces are crucial for Java to be Java. The Java Core APIs provide the basic language, utility, I/O, network, GUI and applet services.

There are certain methods in the Core API that must be called or implemented in a certain pattern. By failing to follow these patterns it is possible to write a program that is syntactically correct, but which will be highly nonportable. Sun's Pure Java Cookbook provides five or so examples and Java Pure Check will catch most of them.

On the flip side of the coin, any implementation of the Java Core APIs will include classes and packages that are not part of the documented API interface. Portable programs must not depend on these implementation details as they may vary between different Java implementations. Sun cautions that this is true even if the classes in question are undocumented parts of its reference Java platform implementation. Those interfaces are not part of the Java platform definition and they are not checked by the tests for Java compatibility so they may be absent or may behave in subtly and dangerously different ways on different Java implementations. They are not documented because they are not intended for client use.

One subtle way that a program may depend on implementation details is by defining classes into the packages that are part of the Core APIs or a specific implementation. This breaks protection boundaries that the Core implementors are entitled to count on. Another subtle dependency on implementation details is direct use of the AWT component peer interfaces defined in classes in the java.awt.peer package. These interfaces are documented as being "for use by AWT implementors". A portable program will use the AWT rather than implement it.

Hardwired, platform-specific constants are a definite portability problem and a violation of Pure Java guidelines. Hard-coded file names should not be used and directory paths tied to a file name are even worse. Similarly, input and output streams can be used unportably with hard-coded and hardware-specific line termination characters. Java Core APIs do provide portable alternatives that should be used.

The most portable way to construct a File for a file in a directory is to use the File(File,String) constructor to build up the path. Other portable solutions are to use the system properties to get the local file separator and starting directory, or to use a file dialog to ask the user for a filename.

"Testing all classes and methods is simply an excellent requirement, especially as hectic development schedules stretch resources and create a temptation to look for short cuts, or as development teams expand."

Various hardware platforms may also have a variety of environmental differences, such as screen size, available fonts or different color pallets. Developers need to look for the most portable way to implement functions that are affected by such variables. Don't hard code text sizes, for instance, and have applications get the font names from the java.awt.Toolkit.getFontList method rather than using a hardwired font list.

Applications vs. Applets

Most developers would agree that writing portable applets is more challenging than writing portable applications. Applets extend the java.applet.Applet class. Portability, however, can be affected by a variety of other factors, including the Web page the applet loads from, the other classes the applet uses, the HTML that loads the apple, and the security manager and AppletContext of the user's browser.

For example, the HTML <applet> tag requires that the applet markup follow certain rules. <param> elements must come before the alternate contents. Alternate contents are text elements, like the contents of a paragraph – the <p> tag. This means no <p> tags can be included in the alternate contents.

Applets will almost certainly have to run under the control of a security manager. There is, however, no standard profile for security managers. The user can instruct their security manager to deny any combination of access. The best answer is for developers to make sure that an applet handles any security exception gracefully.

Similarly, the Java standard does not specify required content types or protocols. The MIME types image/jpeg and image/gif should be safe, as are the http:, file: and ftp: protocols. Developers should again ensure that the applet will handle any errors gracefully.

And finally, the Java 1.0 API specification of the AppletContext/Applet protocol was not very precise. The result of this ambiguity is that different browsers call the applet's enter and leave methods at different times. The protocol specification is much more precise in the Java 1.1 API, so this problem will disappear in time. Until Java 1.1 is universally deployed, developers should ensure that an applet that behaves politely on one browser does not hog resources on another, or that an applet that functions normally on one browser does not stall on another.

Final Notes

Developers charged with creating Pure Java applications should certainly read Sun's guidelines and its cookbook for creating portable Java applications. There are also various online discussions of bugs and other factors that sometimes make universally portable Java code more challenging than developers desire. In addition to Java Pure Check, I also use the rest of the Java quality assurance tools developed by Sun's SunTest business unit. These include JavaStar for GUI testing and JavaSpec for API testing.

Development teams must, in addition to employing good testing tools, clarify their expectation for Pure Java coding and make sure all team members understand what that means. With the proper foundation, an emphasis on delivering Pure Java code wherever possible can offer substantial benefits to the developer.

About the Author

Bob Adams is Director of Business Development for Cupertino, California-based SoftPlus.He can be reached at boba@softplus.com

boba@softplus.com

Schlumberger to Provide Smart Cards, Terminals to Visa Financial Institutions

(New York, NY) - Schlumberger has signed a letter of intent with Visa International to become a partner in the Visa Smart program, which offers complete smart card solutions to Visa financial institutions worldwide.

As part of the program, Schlumberger will provide cards, terminals and application development support and services for added value programs, such as loyalty, to Visa Members. This offering will include the whole range of Visa Smart Products and the Open Platform, a secure multi-application card based on Java™ Card technology, as well as the company's MagIC™ range of technology-leading point-of-sale terminals.

The VisaSmart program provides Visa Members with access to Schlumberger's full range of single application and multi-application smart card products, technologies and services conforming with Visa Chip applications standards.

For more information on Schlumberger Electronic Transactions, see their Web site at www.slb.com/et.

Visionary Solutions Introduces VisImage® JAVA Imaging Class

(Philadelphia, PA) - VisImage® JAVA Imaging Class, written in 100% Pure Java™, is now available from Visionary Solutions, Inc. This product offers a full range of imaging, document management and document cleanup features and can be used with any 4GL Java applications development tool such as PowerJ, JBuilder, Symantec Café and Visual J++.

The VisImage® JAVA Imaging Class is distributed as a JAR file with a size under 120K. It is available for purchase for $795.00. For more information, call 215 342-7185, Fax 215 728-1134, e-mail visolu@voicenet.com or see their Web site at www.visolu.com.

WebLogic Announces Intel Partnership

(San Francisco, CA) - WebLogic, Inc. has announced that Intel Corporation has made an investment in WebLogic.

WebLogic also announced that Intel and WebLogic will work together to optimize the performance of WebLogic's Tengah Java™ application server running on the Intel Architecture, including the future IA-64TM product family, the first of which is code-named the Merced™ processor. The Merced processor is scheduled for production in 1999.

Information about WebLogic's products, services and strategic technology partners can be found at their Web site at www.weblogic.com or by calling the company at 415 659-2600.

IBM Licenses Sun's picoJava Processor Design

(Fishkill, NY and Palo Alto, CA) - IBM and Sun Microsystems, Inc. have announced that IBM has licensed Sun's picoJava I processor core. picoJava I is a microprocessor chip design that can be used in consumer electronic products such as cellular phones, TV set-top boxes and other information appliances, to accelerate Java™ application performance. This will allow manufacturers to provide new types of services on smaller, easier-to-use devices.

With picoJava, IBM can build microchips with support for Java software embedded directly on the chip. This approach will help provide fast, efficient operation of Java applications on small electronic devices that are less powerful and have limited memory capacity compared to desktop computing systems.

Information on IBM Microelectronics products and services can be found at www.chips.ibm.com.

ObjectSpace Selected by Sun for New Java™ Consumer Alliance

(Dallas, TX) - ObjectSpace, Inc., a leader in providing solutions to the distributed computing marketplace, has announced its charter membership in the Java Consumer Alliance Program hosted by Sun Microsystems, Inc.

Under the alliance, ObjectSpace will continue to provide software development and integration services to companies while focusing on three key areas:

- Embedding the PersonalJava platform into consumer devices with compact information displays such as Web phones and hand-held computers
- Leveraging the embedded Java platform to assist in the delivery of wireless and manufacturing devices such as cellular phones and computer-aided manufacturing systems
- Continuing its emphasis in developing highly distributed systems.

For more information on ObjectSpace, visit their Web site at www.objectspace.com.

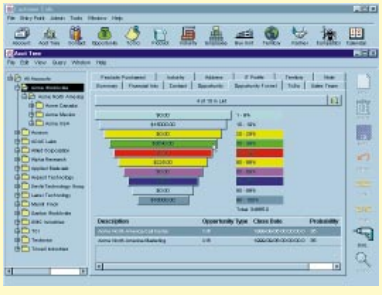
Stingray Software Announces Objective Toolkit/X 1.0

(Morrisville, NC) - Stingray Software Corp. is shipping Objective Toolkit/X, the first docking form for Visual Basic. Objective Toolkit/X allows the Visual Basic developer to enhance their forms so that they can dock or float any MDI child. Docking forms let developers double-click to quickly change between an MDI child form and a docked or floating form.

For more information about Objective Toolkit/X, see their Web page at www.stingray.com/otx/default.asp, call 800 924-4223 or Fax 919 461-9811.

Sales Vision Introduces Customer Café™

(Charlotte, NC) - Sales Vision, Inc. has announced the introduction of Customer Café, an SFA solution based entirely on Java™. Customer Café contains 20 business objects which represent each facet of the selling environment; i.e. account management, team selling, forecasting, etc. Individual objects can be incrementally layered into the application to reflect the customer's unique business vision. This highly adaptive development approach allows customers to rapidly assemble and deploy customized solutions to accommodate their evolving business requirements.



Customer Café includes all Java source code classes and can be maintained in any "Bean-enabled" development environment. All major relational databases, including Sybase, Oracle, Informix and Microsoft SQL Server, are supported.

Customer Café is priced at $1,850 per user. For more information, visit the Sales Vision Web site at www.salesvision.com, call Mark Logan at 704 643-1000 or fax 704 643-1090.

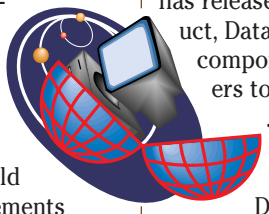
Sales Vision

JDJ NEWS

JDJ Bundles Visual J++ Technology Preview 1

(San Francisco, CA) - SYS-CON Publications, Inc., recently bundled and distributed Microsoft's newly announced Visual J++ 6.0 Technology Preview 1 with ***Java Developer's Journal***.

The magazines were available during the recent conference. "We will be distributing 15,000 bonus copies of ***Java Developer's Journal*** at JavaOne," said Fuat Kircaali, publisher of ***JDJ*** and president of SYS-CON Publications, Inc. "The special issue of ***JDJ***, with a record circulation of 65,000 copies, which includes world exclusive announcements and giveaways, hit newsstands on April 7, 1998 and is expected to sell out."



"We exclusively chose ***Java Developer's Journal*** to introduce Visual J++ 6.0 to the Java community because of its vendor-neutral policy, high circulation and highly respected position in the Java media," said Farhana Ahmad of Microsoft. "Microsoft Visual J++ 6.0 Technology Preview 1 is the easiest way to harness the productivity of the Java language and the power of Windows to build high-performance, feature-rich applications and components."

For more information, press only, contact Sue Lindsey at 425-637-9097 or e-mail suel@wagged.com.

JDJ Bundles IBM Product Demos at JavaOne

(San Francisco, CA) - SYS-CON Publications, Inc. recently bundled and distributed a sampler of IBM products with ***Java Developer's Journal***.

The magazines were available during the recent conference. Included on the disk are a number of third-party product demos as well as IBM products.

The IBM products included are San Francisco, ServletExpress Beta 2.0, Visual Age for Java Entry and WebRunner Bean Tools. The third-party products included are Activerse Ding!, Blue Lobster Stingray and Mako Server, InstallShield Java Edition Version 2, iRenaissance Calendar Central, Marimba Castanet 2.0 and Orbital Technologies Organik.

Protoview Development Builds on JFC with Data ExplorerJ

(San Francisco, CA) - ProtoView has released its first JFC product, Data ExplorerJ. This component allows developers to display and edit Java™ application data in the Windows Explorer UI.

Data ExplorerJ is one of the first products to fully take advantage of the Java Foundation Classes (JFDC). It integrates JFC components – JTree, JSplitter and JTable.

Data ExplorerJ is shipping now and retails for $299 ($499 with source code). For more information, contact ProtoView at 800 231-8588 (609 655-5000 for international orders) or see their Web site at www.protoview.com/dataexplorer.

Halcyon Introduces Instant Basic for Java

(San Jose, CA) - Halcyon Software, Inc. has introduced Instant Basic™ for Java which allows developers to both migrate existing Visual basic applications to the Java platform and to create new Java-based applications.

Instant Basic™ for Java will be available in both Standard and Professional editions. The Professional edition is bundled with the Instant Converter and the Instant Installer™. It also includes Professional Controls, ActiveX support (Windows only), DAO/RDO support with JDBC and is bundled with an evaluation version of Cloudscape's JBMS™.

The Standard edition is priced at $99; the Professional edition is $795. For additional information, call 408 998-1998 or visit Halcyon's Web site at www.halcyonsoft.com.

Object People Releases TOPLink™ for Java

(Ottawa, CAN) - The Object People has released TOPLink for Java. Since Java is an object-oriented language and most databases are relational, TOPLink solves the problem of having them work together by mapping Java objects to relational databases. Developers can save 25-40 percent of the time usually required to get their applications to work.

TOPLink incorporates features such as object-level transactions and queries, allowing developers to work exclusively with objects. No SQL programming is required.

For more information on TOPLink for Java, visit their Web site at www.objectpeople.com.

KL Group Releases Version 3.0 of JClass Products

(Toronto, ONT) - KL Group, Inc., a leading provider of GUI components and Java development tools, has upgraded its entire family of JClass products to version 3.0. This version features compatibility with the JavaSoft "Swing" component toolkit in the JDK 1.1 and the forthcoming JDK 1.2. It offers a synchronizing release of JClass BWT, Chart, Field and LiveTable, ensuring that developers can easily identify compatibility with the latest JDK.

For more information on this release, see www.klg.com.

Added Value for E-Commerce, Expanded Java & Security Features

(Sebastopol, CA) - WebSite Professional 2.2 is now available from O'Reilly. This version includes Uplink, O'Reilly's new utility designed for Internet Content Providers and Internet Service Providers. It also has enhanced log file management and generation, more options for Java servlet development and includes Live Software's new JRun 2.1.

Suggested list price for WebSite Professional 2.0 is $799; the upgrade to version 2.2 is free for downloading at website/oreilly.com by registered version 2.0 and 2.1 customers. For more information, call 707829-0515 or e-mail order@oreilly.com.

AlphaBlox Corp. Ships Flagship Product

(Orlando, FL) - AlphaBlox Corp. is shipping AlphaBlox Enlighten, the first complete system that provides easy-to-use Operational analysis applications for line-of-business users via the Web.

AlphaBlox Enlighten consists of three parts. Ready-to-use Java Building Blox can be assembled into analysis applications without coding. InterBlox™ is the Dynamic Application Assembly framework that handles communications, cooperation and control of the Blox. BASE is an extensible server environment that enables IS centralized maintenance and administration of AlphaBlox Enlighten applications. It also comes with two ready-to-run applications that are examples of what customers can assemble.

Release 1.1 is available now by calling 888 BLOXNOW. Pricing starts at $50,000 per server for up to 50 users. For more information you may also see their Web site at www.alphablox.com.

Rogue Wave Software Announces Java Products

(San Francisco, CA) - Rogue Wave Software, Inc, announced three new products at the JavaOne™ Conference. Grid.J 2.0 is a major update to the Grid/J product from its Stingray Software Division. It features full formula support to turn a grid into a fully-functioning spreadsheet. Blend.J 2.0 integrates Rogue Wave's JWidgets and Stingray's Blend product into a collection of more than 25 controls for Java developers. StudioJ is a new suite of Java components that combines technology from both company's for Java component development.

North American pricing for Grid.J 2.0 is $395; it ships with full source code and 30 days of technical support. Blend.J 2.0 is priced at $295, and also ships with full source code and has 30 days of technical support. For more information call Rogue Wave at 303 473-9118, e-mail websales@roguewave.com or visit their Web site at www.roguewave.com.

Sun and IBM Team to Accelerate Adoption of JavaOS™

(Palo Alto, CA and Somers, NY) - Sun Microsystems, Inc. and IBM have announced that they are collaborating to deliver JavaOS for Business™ software, a new operating system software product optimized for network computing on the Java™ platform. This alliance will accelerate the adoption of a more affordable, easier to manage and simpler model of computing.

The two companies will jointly develop and co-market the JavaOS for Business software, which will provide computer and component manufacturers, software vendors, channel integraters and enterprise customers with an open industry platform optimized to run Java applications in a centrally managed environment. It is designed so client machines can connect to any platform and be centrally managed from a wide variety of server platforms. It should be available to manufactures in mid-1998.

Additional information on Sun Microsystems and IBM can be found on their Web sites, www.sun.com or www.software.ibm.com.

Symantec Offers Free JDJ Subscription with Purchase of Visual Café

San Francisco, CA) - Visual Café for Java Professional Development Edition and Database Development Edition are available immediately from Symantec Corp. Visual Café for Java is their open standard, extensible Rapid application Development Java software development tool for writing, debugging and deploying platform-independent Java applets and applications.

Included with Version 2.5 is a coupon for a complimentary one-year subscription to ***Java Developer's Journal.*** Mail in your copy of the receipt for purchasing Visual Café 2.5 for Java and you will receive your free subscription.

For more information on Visual Café 2.5, call 541 334-6054 or see Symantec's Web site at www.symantec.com. For more information on this special subscription offer, see cafe.symantec.com/promo/jdj.

Live Software Will Support Borland Products

(San Francisco, CA) - Live Software has announced ServletDebugger 2.0, an enhanced version of its acclaimed servlet support tool, for Borland International Inc.'s JBuilder2, the new version of Borland's award-winning family of visual development tools for building corporate and enterprise software applications with the Java™ programming language.

ServletDebugger 2.0 allows for the testing and debugging of Java servlet code from within the JBuilder development environment.

ServletDebugger 2 is available from Live Software's Web site at www.livesoftware.com and is priced at $195 per developer seat. Additional information about Borland International may be found at www.borland.com.

InnoVal Promotes Java Lobby

(Harrison, NY) - InnoVal systems Solutions has launched an advertising campaign to encourage OS/2 users and developers to join the Java Lobby. The Java Lobby is an advocacy group that represents the needs and concerns of Java developers and users throughout the world.

InnoVal released the beta version of J Street Mailer at the end of January. Following the test period, they intend to market J Street Mailer as a full function e-mail client and a module that can be integrated into other Java applications that require e-mail functions.

For more information about InnoVal and J Street Mailer, see their Web page at www.innoval.com. More information on the Java Lobby may be found at www.javalobby.org.

DashO-Pro Increases Speed & Decreases Decompilation

(San Francisco, CA) - preEmptive Solutions, Inc. has announced DashO-Pro. DashO-Pro increases Java program speed, makes programs as small as possible and helps protect Java programs from decompilers, all while maintaining platform independence. Although it cannot prevent decompilaton, the technology makes the resulting reconstructed source extremely difficult to understand.

DashO-Pro is available now at a special introductory price of $1,495 directly from preEmptive by phone at 216 732-5895 or from their Web site at www.preemptive.com.

Novera™ Introduces New Java Application Server

(Burlington, MA) - Novera™ Software, Inc. has announced the most complete, best managed, Java application server as the foundation for the company's new jBusiness™ Solutions.

jBusiness Solutions are about Java-enabling the core business process to improve the quality of development, reduce cycle time and get more results from limited resources.

Pricing for the Novera application and Management Servers start at $9,995 per server for Internet capabilities and at $200 per user for intranet capabilities. For more information, call 781 270-4422 or visit their Web site at www.novera.com.

JDJ Sponsors Java Pavilion at iEC

(Pearl River, NY) - ***Java Developer's Journal*** is sponsoring the Java for iEC Pavilion at the Internet and Electronic Commerce Conference and Exposition at the Jacob Javits Center in New York on April 27-29. iEC is the nation's largest event dedicated exclusively to promoting the use of Electronic Commerce via the Internet.

The Java for iEC pavilion is a specially designated area near the front of the iEC exhibit hall which will spotlight Java developers demonstrating iEC applications of their software.

For more information on iEC, see their Web site at www.iecexpo.com. For more information on ***JDJ,*** see www.javadevelopersjournal.com.

Hit the Road, Joe

THE GRIND

by Joe S. Valley

I have been hinting that things are not going well at work. Yeah, the end is near for old Joe. Projects on hold all of a sudden, my boss in shouting matches with the president, the guys on the Board of Directors coming and going a lot. Yep, something is going on.

Moving to this company was a risk from the start. But, you've got to play the game. The pay here is very good, the people are OK and the projects are cool. I can't really complain. I'll add the stock option paper to my recycling bin, as I have with most small companies where I have worked. Options are great, but don't count on them to pay the rent.

The average length of a job in The Valley is 18 months, so I am right on schedule to be gone.

Like my late mystic friend Nostrajava, I have developed a sense of the future for any company. All you need to do to determine the fate of a company is to make sure you interview the president before you accept the job. Doesn't matter if you are a VP or a humble engineer – try to meet with the president. In my case, I did, and was troubled afterwards.

The president of my company is young, got his MBA from the big university in Palo Alto. Smart guy, very glib, with the ability to drill down through the BS quickly. All in all, a star player. So what was the problem? Old Joe asked him the killer question.

"Sounds like you have a great background to lead this company. By the way, what was your biggest business failure?" I said towards the end of our brief interview.

He stared at me blankly for a second, but quickly recovered. "Well, I haven't had any failures, that's why I am here!" he confidently stated. It helped that his uncle was a very well known venture capitalist and his father was connected to New York bankers, but I wasn't going to burst his bubble. All I had to do was read *Red Herring* to find out all kinds of things about his background.

So what's wrong with his statement? Everyone has failures or setbacks. It isn't what happens to you, it's how you react to it. If you have glided through life and don't hit your first speed bump until you are a 36 year old president, then the fall will be ugly for you and those around you. The prez is under pressure, and doesn't understand how he will react to this kind of pressure. He may be out soon, but there will be the first quarterly loss, the stock will drop like a rock and everyone will forget about you. The loss came, the stock dropped and now we are slicing and dicing budgets. I will be one of the first to go; I'm over 40 and expensive. We don't need wisdom and experience here. We need results... by next quarter.

Despite his overconfidence, the president is a good guy. Laying people off is not something that he wants to do. Now is my chance to show a little leadership. I knock on his door.

"John, the caca is flying, people know something is happening. We will need to cut headcount. By the way, have you ever laid anyone off before?" I ask.

"Uh, no," he replied, looking a bit shell-shocked. "But I can handle it, don't worry," he fires back in his best sales voice.

"John, you're a smart guy, but listen to me on this. I have been on both ends of layoffs. Do them quickly!. Get your managers together tomorrow. Draw up a list of people you cannot afford to lose. Tell them you need them to stay, and double or triple their stock options. Next, figure out who else you want to keep and have their managers talk to them; tell them they have jobs. Next, draw up the layoff list. Do it by Friday. Have HR work over the weekend to get the packages together. Monday afternoon, have the managers lay off the people; you do the Vice Presidents and Directors. Do it right after lunch and send everyone home. Tuesday morning have an all-hands meeting to talk about what you are going to do to turn the company around. Don't dwell on the layoffs. Take responsibility for the problems, then quickly tell everyone what the company is going to do to turn itself around. Throw everyone a bone, like two extra vacation days or a hundred shares of stock. Everyone, including the facilities guys. The most important thing is to move quick and nail down your best engineers and top salesguys. If they split, you can't turn the place around. The headhunters are circling this place now. My best coder got three calls yesterday!"

He stares at me for a second. "Sage advice, Joe. You know, the biggest problem with being the president is that you have no friends. People aren't predisposed to be honest with you, even your board members. You are the first honest person I've talked to today!"

"Thanks John," I stop for a second. "You have the hardest job in the company. And remember, everyone here wants you to succeed."

I took a deep breath. "In the spirit of honesty, you are going to have to take out me and my entire team; we both know my project is doomed. I'll have my guys get their resumés together soon. Give them a decent layoff package, as they work hard. A reference letter from the VP of Engineering would be nice."

"You got it," John says in a far away voice. "Thanks, Joe. By the way, what will you do?"

"Hey, I have my writing gig, and I need to get that '57 Chevy in my garage back together. Ever rebuild a small block Chevy engine? Great therapy. Anyway, I can always scrape up a couple of days a week of contract programming." I'm really not worried. The job market is hot and I'll find something within a month if I want to.

Such is life in The Valley.

"The average length of a job in The Valley is 18 months, so I am right on schedule to be gone."

Joe S. Valley is a scarred veteran of the Silicon Valley wars. It was either writing this column or heading back into therapy. His company can't afford mental health care coverage anymore, so writing is the only option. There are a million stories in the Valley and Joe knows lots of them. Got a good story? E-mail him at Joe@sys-con.com

Joe@sys-con.com

Ad

KL Group
Full Page Ad