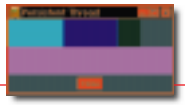
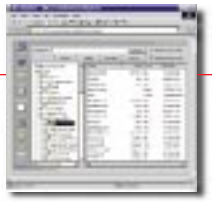
JAVA™ DEVELOPER'S JOURNAL
JavaDevelopersJournal.com
Volume: 4 Issue: 2, 1999

7 NEW JAVA PRODUCTS REVIEWED

SYS-CON PUBLICATIONS





JAVA DEVELOPER'S JOURNAL

EDITORIAL ADVISORY BOARD

Ted Coombs, Bill Dunlap, David Gee, Michel Gerin, Arthur van Hoff, Brian Maso, John Olson, George Paolini, Kim Polese, Sean Rhody, Rick Ross, Ajit Sagar, Richard Soley

Editor-in-Chief: Sean Rhody
Art Director: Jim Morgan
Executive Editor: Scott Davison
Managing Editor: Hollis K. Osher
Senior Editor: M'lou Pinkham
Production Editor: Brian Christensen
Technical Editor: Bahadir Karuv
Visual J++ Editor: Ed Zebrowski
Visual Café Pro Editor: Alan Williamson
Product Review Editor: Jim Mathis

WRITERS IN THIS ISSUE

Jim Barnebee, Brent Callaghan, Tim Callahan, Andrei Cioroianu, Claude Duguay, Moe Fardoost, Anil Hemrajani, Rob High, George Kassagbi, Gabor Liptak, Qusay H. Mahmoud, Jim Milbery, Sean Rhody, Robert Tiffany, Alan Williamson, Ed Zebrowski

SUBSCRIPTIONS

For subscriptions and requests for bulk orders, please send your letters to the Subscription Department

Subscription Hotline: 800 513-7111

Cover Price: $4.99/issue
Domestic: $49/yr. (12 issues) *Canada/Mexico:* $69/yr.
Overseas: Basic subscription price plus airmail postage (U.S. Banks or Money Orders). *Back Issues:* $12 each

Publisher, President and CEO: Fuat A. Kircaali
Vice President, Production: Jim Morgan
Vice President, Marketing: Carmen Gonzalez
Advertising Assistants: Robyn Forma, Jaclyn Redmond
Accounting: Ignacio Arellano
Graphic Designers: Robin Groves, Alex Botero
Webmaster: Robert Diamond
Customer Service: Sian O'Gorman, Paula Horowitz
Online Customer Service: Mitchell Low

EDITORIAL OFFICES

SYS-CON Publications, Inc.
39 E. Central Ave., Pearl River, NY 10965
Telephone: 914 735-7300 Fax: 914 735-6547
Subscribe@SYS-CON.com

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

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

SYS-CON PUBLICATIONS

FROM THE EDITOR

Sean Rhody, Editor-in-Chief

Java and All That JiBE

Java Developer's Journal was among the many exhibitors at the Java Business Expo at the Jacob Javits Center in New York City. I was only able to make it for one day, but I managed to pack a great deal of interviewing, observation and conversation into that day, in addition to presenting our Editor's Choice Awards. (See ***JDJ*** Vol. 4, Issue 1.)

Probably the biggest news from the show was the renaming of the JDK 1.2 and its accompanying APIs to Java 2. This move is a signal from Sun that some changes are happening with Java, and that now is the time to take notice. One important change is in the way other vendors will license the Java platform. The so-called "Cold Room" vendors – those who develop to the specifications from Sun, but who don't use their code – may find it easier to come in from the cold with the new licensing. Additionally, Sun has relaxed on the requirement that innovators assign back to Sun the rights to all improvements. Grumbling from vendors at the show indicated that perhaps these moves were not enough, but only time will tell.

Interestingly enough, these moves may prove disastrous for Java. Why? Look at the SQL standard, then try to find one vendor who fully supports it. Each database vendor has its own proprietary, value-added version of SQL. Granted, the early SQL specifications lacked enough detail for procedural language, forcing the vendors to create their own, but as a result, you can't write a stored procedure in Oracle and expect it to run in an SQL Server or DB2. While the Java specifications are much more detailed, the license changes may allow for such differentiation. It'll be interesting to see how these changes are accepted and implemented.

I had a chance to have a good long talk with the IBM guys concerning their Application Server offerings. IBM announced their WebSphere Enterprise edition, which integrates EJB support into their WebServer offering. This announcement is less compelling than the overall story that IBM can tell in the Server arena. What they offer that's unique is the ability to interoperate with a large variety of clients, servers and operating systems. Add to that database support on any platform you care to name, asynchronous messaging through MQSeries, and platform-specific native compilers that turn byte code into native code, and you have a potent offering. Overall an evolutionary – rather than revolutionary – product offering, but attractive nonetheless, especially to those businesses with a significant investment in IBM technology.

BEA was another server vendor that I spent time with. BEA recently acquired Weblogic and has renamed the company BEA/WebXpress. WebLogic was one of the first EJB servers, and with the addition of the expertise of the BEA Tuxedo staff – particularly the expanded support organization – WebLogic looks to be one of the servers that'll survive the inevitable thinning of this particular field.

I also stopped by the NuMega booth for a look at some of their tools. They've developed a particularly interesting thread monitor that can identify deadlocks and other thread-related errors that are particularly difficult to track down. The tools only run under NT, but since most development is done there, NuMega feels that it's not that big of a disadvantage. There are also a number of diagnostic and testing tools appearing on the market, which should provide some relief to those of us trying to figure complex programming errors out with println statements.

I'm sorry that I had only a single day to spend at the JiBE show. Sun, as befits their position as the parent of the whole movement, had a huge set-up that I didn't get a good chance to explore. On the whole though, I found that the two biggest topics of the show were the Java 2 release and the rise of the EJB Application Server. I look for this to be an interesting year for development.

About the Author

*Sean Rhody is the editor-in-chief of **Java Developer's Journal**. He is also a senior consultant with Computer Sciences Corporation where he specializes in application architecture, particularly distributed systems. He can be reached by e-mail at sean@sys-con.com.*

GUEST EDITORIAL

Rob High

The Synergy of Java and CORBA

When I began using CORBA in 1993 I was impressed by how well (and easily) I could define an object model and express it to other developers. That, of course, is made possible through CORBA Interface Language. More fundamentally, it is achieved with a strong and inherent distinction between interface and implementation – a distinction that's often lost in programming languages. CORBA IDL is absolutely and only about interface.

After designing an object model based strictly on interface semantics, I can ensure that only the interface semantics forms the contract for other users of my objects. My users avoid building any dependencies on my implementation semantics – thus increasing both the portability of their applications and the reusability of my objects.

The other thing distinctly impressive about CORBA is its ability to scale. CORBA is about architecture and specification of programming models that are, at their very core, designed and intended to be deployed and to interoperate across distributed systems. On one hand, the CORBA specifications are focused on programming and deploying applications in a distributed system, but don't define the qualities of service supplied by any given implementation. On the other, these same specifications don't inhibit, and in fact specifically enable, high levels of quality service in particular implementations. This enables different vendors to provide lightweight technology packages for small-scale deployments and robust platforms for large, enterprise-scale deployments. The same fundamental programming model can be used across all of these deployments from one end to the other -- mostly.

I encountered Java for the first time three years ago at a CORBA conference – I was presenting the merits of CORBA, and everyone else was presenting the merits of Java. I was overwhelmed with the hype and not just a little skeptical about its applicability and durability. Yes, Java seemed to introduce fundamental productivity gains, and yes, the idea of "write once, run everywhere" was enticing, but how could a garbage-collecting interpretive language scale? Moreover, the examples of its use were mere toys -- silly demonstrations of bouncing icons and checkers games. And, of course, there was the meteoric rise in interest in Java (remember Tickle-Me Elmo?)

But there's important synergy between Java and CORBA. Essentially, it comes down to this: Java, and more particularly Enterprise JavaBeans, provides a fully consistent programming model that can be moved between platforms supporting different levels of scalability. CORBA provides the definition of infrastructure and services that enables different levels of scalability in a distributed system. EJBs close the gap between mostly transparent and completely transparent scalability. This is achieved by removing business object developers from any dependency on the quality of service of the underlying platform. EJB developers concentrate on programming business objects that encode business function rather than information technology and infrastructure function. CORBA enables complete location transparency, so that as an application or enterprise grows, its infrastructure can be scaled up as the EJB implementation remains the same. Moreover, since EJBs don't encode quality of service assumptions, the qualities of service can be increased (or decreased) based on the evolving requirements of the business.

Java and EJB define specific mappings to CORBA services for Naming, Security and Transactions, and in both directions between Java interfaces and CORBA IDL. The quality-of-service semantics that you can specify as EJB descriptors – including the transaction management descriptors such as TX_REQUIRED and TX_SUPPORTS as well as the run-as security descriptors such as CLIENT_IDENTITY and SYSTEM_IDENTITY – map well to semantics that are easily implemented using CORBA services and quality-of-service policies attributed to CORBA Object Adapters. As a result, CORBA and Java work well together – by design.

Java gives you portability (write once, run everywhere), not just across heterogeneous platforms as it always has but – when combined with CORBA – across different qualities of service and scaling. CORBA provides a common, consistent and interoperable specification for distributed object management and thus enables differing qualities of service and scale on which EJBs can execute. An enterprise can pick and choose a platform for its quality of service and for how well it fits into their overall computing requirements and budget, rather than on the programming model constraints it might impose. Application developers can increase productivity through the use of Java language and its corresponding Web-enablement features. They're freed from infrastructure issues and can concentrate on providing business value. Conversely, enterprises obtain the flexibility to deploy their applications in a way that fits their integrity and scalability needs gives them the interoperability assurance provided by CORBA.

About the Author

Rob High Jr. is a senior programmer in IBM Component Broker Architecture and Development. He can be e-mailed at rhigh@austin.ibm.com.

PERSISTENT Threads

PART 2

When it comes to event preservation, a persistent thread is worth a thousand objects

by **Andrei Cioroianu**

Persistence is our way to fight the decay of time. We take pictures and film events in order to remember, review and analyze them. We freeze perishable products in order to preserve or transport them over long distances. And in much the same way, computer users save ideas and programs as files on hard disks and transmit them over networks so that they too can be printed and preserved – persisted – over time.

The same technology used to preserve an event (such as a camera taking a picture), can be used to broadcast that event live, over a network, assuring the event's persistence. And with computers, the Serialization API of the Java platform is used to implement the object persistence or to send objects via Remote Method Invocation (RMI).

In a previous article (***JDJ*** Vol. 3, Issue 8), I showed how object serialization might be used to create persistent user interfaces (Reference 1). No threads were created and all the classes used were serializable. However, many Java API classes aren't serializable (i.e., they don't implement java.io.Serializable). One of them is java.lang.Thread.

Why Isn't java.lang.Thread Serializable?

The two main reasons why the Thread class isn't serializable are the dependence of Java threads on the implementation of the Java Virtual Machine (JVM) and the interaction between the concurrent threads. Respecting "The Java Virtual Machine Specification" (Reference 2) and obtaining the best performances when the Java code is run become the main concerns of JVM's vendors.

Some JVMs are better than others because JVM's Specification is flexible, allowing the vendors to adopt different solutions. For example, the stack associated with a thread may or may not be continuous and its size may either be fixed or can change dynamically. In addition, the threads execute a lot of native code (the Java API has many native methods). When this happens, the PC register (program counter) associated with the thread has an undefined value. Hence, the state of a thread isn't always accessible – even to the Java Virtual Machine.

Even if a thread doesn't execute native code, it's still difficult (if not impossible) to detect at runtime all the objects accessed in the thread's code. Note that these objects are stored in the JVM's heap (which is shared between threads.) Some might have their locks locked by another thread. When a thread waits for another thread to release the lock of an object, other threads may acquire the locks of other objects. Using synchronization – without a correct analysis of who might be waiting for who – can lead to deadlock. Usually this analysis can't be made automatically. The communication between concurrent threads through the shared memory is another obstacle for the implementation of the thread persistence at the JVM level.

These are only a few technical reasons why the instances of the java.lang.Thread class can't encapsulate the state of the threads from the JVM's point of view. Therefore there's no point in making java.lang.Thread serializable. This doesn't mean that there aren't other ways to implement thread persistence, however.

What Thread Persistence Actually Means

"The Java Language Specification" (Reference 3) defines a thread as "a single sequential flow of control" within a program. To control the flow, the JVM creates some data structures (the stack, PC, etc.) for each thread. There may also be native resources associated with each Java thread (such as a native thread). The programmer has limited control over a thread through an instance of the java.lang.Thread class.

Each thread has a run() method that's called from the start() method of the Thread object associated with the thread. After the start() method is called, the thread becomes alive. A thread dies when the run() method returns the control to its caller. The System.exit() method kills all live threads.

To be able to define what the persistence of an entity (such as an object or a thread) means, you must know what the state of that entity represents in the moments when this state must be saved. For example, the state of an AWT component consists of the set of the values of its properties (dimension, colors, font types, labels, etc.). Although the AWT components have native peer classes, they can be serialized because the properties that define the state are member variables. Unlike AWT components, the behavior of a thread isn't standard because it's given by the code of the run() method. It's the developer's task to identify the variables accessed within this code that define the state of the thread at a given moment. Implementing the persistence of a particular thread means:

1. To define the state of that thread from your point of view, not JVM's;
2. To be able to save its state when the thread is interrupted;
3. To be able to restart the thread and restore it to its previous state prior to being interrupted; and
4. The result of the thread execution must be the same whether the thread was interrupted or not.

What Is Thread Persistence Good For?

Thread persistence is essential for mobile agents, which basically are objects that can move from one host to another within a network. They're very similar to applets in that they run within a container and have one or more sets of init(), start(), stop(), destroy() methods (or equivalents).

They also possess a moveTo (URL destination) method which calls the stop() method(s), serializes the Agent object (or its equivalent(s)) and dispatches the bytecode and the state of the agent to the destination host. The agent is then deserialized by the container of the destination host, which calls the start method(s). It's the agent developer's responsibility to override the start() and stop() methods that must preserve and retrieve the execution state, respectively. This means that the programmer must implement the persistence of the

agent's thread(s). For more information about agents, see "Design of Multi-Agent Programming Libraries for Java" (Reference 4).

Mobility is the main feature of mobile agents. Other interesting, optional facilities are communication through messages, the means to clone and join agents, and even artificial intelligence. This is why the agents need a special container. If you only need to move tasks to dynamically balance a distributed system, then you may implement mobility in ordinary applications that use RMI or sockets to transmit the current state of mobile tasks (or threads).

The applications, which run for a long time to accomplish a certain task, may also benefit from thread persistence. These applications can become more friendly if they allow the user to suspend them and to resume the task after an undefined period of time. The state of the threads can also be saved automatically from time to time so the user doesn't have to start all over again after a crash.

In all three above scenarios, the programmer's main task is to implement the persistence of the threads. I will give you an example of how to do that.

The PersThread Application

To be able to focus on thread persistence I chose a simple example. The PersThread application opens a window that contains a Close button and a canvas object. The application uses a counter, initialized with 0, which is incremented every 100 milliseconds until it reaches MAX_COUNTER == 300. The counter's value is used to compose a color like this:

```
Color c;
if (counter < 100)
    c = new Color(counter/100f, 0, 0);
else if (counter < 200)
    c = new Color(1, (counter-100)/100f,
0);
else
    c = new Color(1, 1, (counter-
200)/100f);
```

The color and its RGB components are shown in the application's window (see Figure 1).

The PersThread class extends java.awt.Frame and implements java.lang.Runnable (see Listing 1). The counter is run in increments within a thread whose run() method is member of a PersThread object. This object encapsulates the state of the thread and represents the window of the application. The state of the application's user interface, the counter's value and the value of the threadStage member variable compose the thread's state. The static member variables of the PersThread instance aren't parts of the state of the thread (they won't be serialized).

The user interface is built in the PersThread() constructor, which sets the background color, the title and the BorderLayout manager of the window. Next it creates and adds a PTCanvas component and a Close button. A PTAdapter object is created and registered as an event listener to the button and the frame. PTCanvas and PTAdapter are inner classes.

The counter variable is given in increments in the computing() method, called from run() (see Listing 1). The limit parameter indicates how many increments the counter must have. After each increment, the canvas is repainted and a 100-millisecond pause is taken.

The run() method is executed in three stages (i.e., it calls computing() three times). The number of the current stage is stored in the threadStage variable. If the

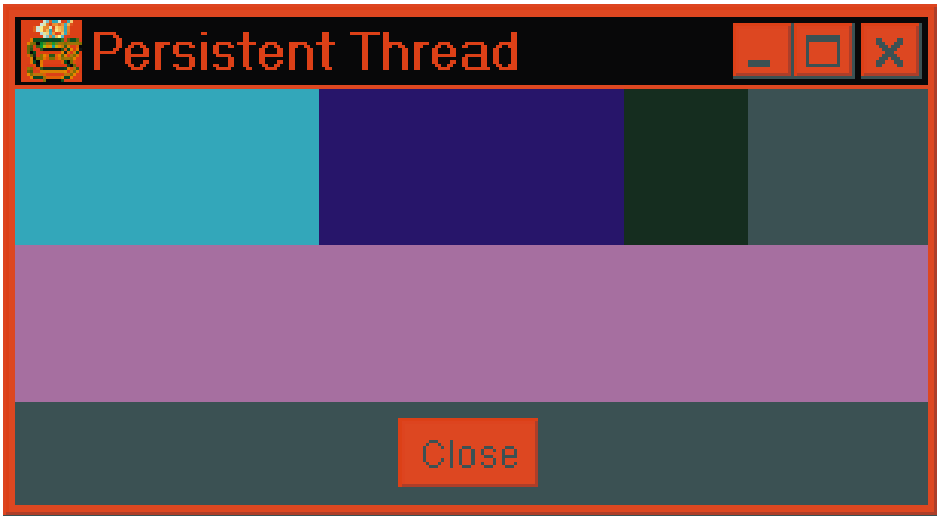


Figure 1 – The user interface of the PersThread application

counter is in the red zone (i.e. 0 <= counter <= 100), then threadStage == 0. If the counter is in the green zone (i.e. 101 <= counter <= 200), then threadStage == 1. If the counter is in the blue zone (i.e. 201 <= counter <= 300), then threadStage == 2.

This section has identified the state of this thread. The next two sections show how to interrupt this thread and serialize its state.

Thread Breakpoints

The run() method calls computing() three times. To simulate a real-world application, I chose not to interrupt the computing() method, whose code may be seen as a sequence that once started, must also be completed. If the Thread.sleep() call throws an InterruptedException, then the mustBeInterrupted flag is set to true. Before returning the control, the computing() method checks this flag. If its value is true, then the interrupt() method of the current thread is called.

The run() method calls breakPoint() between two computing() calls (see Listing 1). The breakPoint() method uses Thread.interrupted() to check if the interrupt() method of the current thread was called. If interrupted() returns true, then breakPoint() calls the serialize() method of the PersThread object to save the state of the thread. Then it calls System.exit() to close the application.

Note that the interrupt() method doesn't stop the thread. It just signals to the thread that it should arrange its own death. You must not use the stop() method (which was deprecated in JDK 1.2) instead of interrupt(), because stop() kills the thread without warning. After a stop() call, the state of a thread might remain inconsistent.

Implementing Thread Persistence

Now you know what thread persistence means and what it's good for. The next question is how to implement the persistence of the thread.

The state of that thread consists of the state of the application's user interface, the value of the counter and the value of the threadStage member variable. All this data, which defines the thread's state, is either contained or referenced directly or indirectly by the member variables of a PersThread object. The serialize() method of this object calls writeObject() from the ObjectOutputStream class to save into a file the state of the thread whose run() method counter is in increments.

```
ObjectOutputStream s = new ...
s.writeObject(this);
s.flush();
s.close();
```

If the Close button is clicked and the thread is alive then the interrupt() method is called (see the appExit() method of the PTAdapter class from Listing 1). Then the Thread.sleep() call from computing() will throw an InterruptedException. The catch of this exception is that it will neutralize the effect of the interrupt() call. However, the mustBeInterrupted flag will be set to true. Before returning the control, computing() will call the interrupt() method again. Therefore, when the breakPoint() method gains the control, Thread.interrupted() will return true. Then breakPoint() will call serialize() and System.exit(). Note that breakPoint() can gain the control only between two calls of computing().

The next time the application is run the main() method will have to restore the state of the thread and restart it. The readObject() method of the ObjectInputStream class will be called to deserialize the state of the thread. If this operation succeeds, then the window of the application will be shown on the screen at the same position with the help of the show() method. The components will have the same state. Then the main() method will create a new thread call its start() method.

```
ObjectInputStream s = ...
pt = (PersThread) s.readObject();
s.close();
pt.show();
thread = new Thread(pt);
thread.start();
```

The start() method calls the run() method of the PersThread instance. To continue the execution of the application from the point where it was interrupted, the run() method uses the value of the threadStage variable. The computing() calls – which were completed at the previous run of the application – are now skipped. For example, if threadStage is 1, then the application was interrupted after the first computing() call (before the counter passes from the red zone to the green zone). The first computing() call is then skipped so the counter doesn't have to go through the red zone again. If the user clicks the Close button when the counter is in the green zone, then the application is interrupted right before the counter passes into the blue zone. If the counter manages to enter the blue zone before the user presses the Close button, then the application can't be interrupted anymore because from this point, no more breakPoint() calls are made.

If the deserialization fails (e.g., FileNotFoundException is thrown), then the main() method creates a new instance of the PersThread class, shows the window and starts the thread.

```
pt = new PersThread();
pt.show();
thread = new Thread(pt);
thread.start();
```

An important observation is that the result of the application's execution is the same, no matter whether the application is interrupted or not (i.e., the white color is shown after 300 increments of the counter). In addition, the execution time is almost the same because the main actions are the same (i.e., the computing() method is completed three times whether or not the application is interrupted).

Synchronization and Inner Classes

The code of the PersThread application is executed within three different threads. The first thread is created by the Java Virtual Machine for the main() method of the application. (The run() method of this thread calls main().) The second thread is created in the main() method. The run() method of the second thread calls comput-

ing() for increments from the counter. I implemented the persistence only for this thread. The third thread is an EventDispatchThread of AWT. It takes events off the EventQueue of AWT and dispatches them to the appropriate AWT components. Most code of the PTAdapter and PTCanvas classes is executed within the third thread.

After the computing() method calls repaint(), a PaintEvent is dispatched to the PTCanvas component. The event is processed and the paint() method is executed within the AWT thread. The paint() method needs the counter's value to calculate the color shown in the canvas of the application's window. The counter member variable is accessed in two separate concurrently running threads (the second and the third). Therefore, the code that increments the counter (in computing()) and the getCounter() method have been synchronized.

Note that in this particular application the synchronized keyword isn't actually necessary because the counter variable is atomic (its type is int), and there's no negative effect if getCounter() is called between the two accesses of the counter member variable from "if (counter < MAX_COUNTER)" and "counter++;". This is a very rare combination of coincidences. For example, if the type of the counter variable had been nonatomic (e.g., long), then the synchronization of the accesses to this variable would have been compulsory. The use of synchronization without reason is a bad idea because it reduces the performance of the application. In this article, my purpose was to show that the persistence could be implemented even for concurrent threads. You usually have to synchronize access to the member variables that are used and modified in concurrent threads. The local variables and the parameters of a method can't be accessed from concurrent threads because their values are stored in the stack associated with the thread that calls the method.

A second important observation is that the value of the counter is used in more than one place in the code of the paint() method of the PTCanvas component. This method composes the color shown in the canvas of the application's window. The counter's value is copied into a local variable because the computing() method can increment the counter anytime. Though the paint() method can access the counter directly (because PTCanvas is inner class), the right procedure is to call getCounter(). There's no way to protect the member variables against the methods of the inner classes. You must be very careful when the methods of a class and the methods of the inner classes are executed within concurrent threads.

"The same technology used to preserve an event (such as a camera taking a picture), can be used to broadcast that event live, over a network, assuring the event's persistence."

Application's Exit

The getCounter() method is also called from the appExit() method of the PTAdapter class. When the user tries to close the application's window or clicks the Close button, an event is generated and passed to one of the actionPerformed() or windowClosing() methods. Each of these two methods calls the appExit() method of the PTAdapter instance that was registered as listener to the application's window and to the Close button. The appExit() method calls the interrupt() method of the thread created in the main() method if this thread is alive and if the value returned from getCounter() is less than MAX_COUNTER. Otherwise, the .ser file created in the serialize() method (when the application was interrupted last) is deleted, before the System.exit() call. This operation allows the application to be restarted with counter == 0 after its execution was completed without interruption.

The System.exit() call from breakPoint() is the simplest way to close the PersThread application, but it isn't the most refined because it kills all threads without notice. If an application must continue its execution after the state of a thread was serialized, then the breakPoint() method must be modified. For example, it could throw an InterruptedException instead of System.exit() call. The run() method of the thread would catch this exception and would use the return instruction in the catch block. From this point, the thread would be dead and the application would continue its execution.

Thread Persistence For Real-World Applications

The PersThread application is simple because it implements the persistence of a single thread. A complex application from the real world might be needed to implement the persistence for more than just a thread. If the threads are independent, then those with low priority might be interrupted and their state might be saved on disk to release memory resources for the high-priority threads. Note that the state of a thread can sometimes consist of huge data structures.

The implementation of the persistence may be very complex for concurrent threads. You must analyze all the situations where a thread might wait for another thread to perform an unlock operation. However, the programmer's task is simplified by the fact that the lock and unlock operations are automatically performed.

There are situations when the persistence of a thread simply can't be implemented. A typical situation is when you may not block a shared resource (e.g., a database, a server, etc.), or, if you block it, then you must unblock it as soon as possible – not after an indefinite period of time.

Summary

The programming technique presented in this article is typically used in distributed systems to move tasks from one host to another. Computing-intensive applications can become friendlier with the help of thread persistence, which allows the user to interrupt the applications and resume them after an undefined time or a crash.

References

1. Andrei Cioroianu, "Persistent User Interface for Multiuser Applications", ***Java Developer's Journal***, Vol. 3, Issue 8, www.javadevelopersjournal.com/
2. Tim Lindholm and Frank Yellin, *The Java Virtual Machine Specification*, Addison Wesley, http://java.sun.com/docs/books/vmspec/
3. James Gosling, Bill Joy, Guy Steele, *The Java Language Specification*, Addison Wesley, http://java.sun.com/docs/books/jls/
4. Takashi Nishigaya, *Design of Multi-Agent Programming Libraries for Java*, Fujitsu Laboratories Ltd.,www.fujitsu.co.jp/hypertext/free/kafka/paper/

CODE LISTING

The complete code listing for this article can be located at **www.JavaDevelopersJournal.com**

About the Author

Andrei is an independent Java developer and writer. You can visit his "(a) Java Developer's Page" at www.geocities.com/SiliconValley/Horizon/6481/, and you're invited to send questions or comments about this article to andcio@hotmail.com.

andcio@hotmail.com

The Price Is Right

Or so you may think – but remember, the devil is in the details

by **Alan Williamson**

Here we are again – a couple of pages telling you how it really is. What an interesting and varied month this has been! And a good month for Java. Lots of different things have been happening.

(Before I begin, I want it known that this is the first column I have ever written using a Mac. I'm sorry, but as they say, "When in Rome...." I'm out of my normal Gates world and have been thrown harshly into this Jobs utopia. I don't know about you, but better the devil you know and all that.... Before I go biting the hand that feeds me, I have to thank Frode Hegland for allowing me a bite of his Apple, for without it this article would not have been possible. Frode, on behalf of a Java nation, we thank you!)

This was the month that Java entered puberty. Like a newborn child finding its feet in a forever-changing environment, a significant milestone was reached. This milestone marks the beginning of a new life for us all. What am I talking about? Java 2, of course. But more on that later.

Trait of the Month

As usual – let's not deviate too much from a tried and tested formula – we'll discuss a personality trait. This month it'll be *relief*. That's the feeling that washes over us when we realize the situation isn't as drastic as we first thought. This was the month we at N-ARY breathed a huge sigh of relief that we had indeed taken the Java route. Let me tell you a cautionary story so others may heed the signs before it's too late.

We've been involved in a major project for the past 14 months. We were approached by a Norwegian startup, Liquid Information Company, to build them their dream. Their dream was to change the way people communicated by tearing down the barriers surrounding conventional communication channels. Build a better e-mail system, build a better newsgroup system, build a better system.

So we took their plans, sat down and designed a complete solution that would be flexible enough to meet the demands of a growing future. In other words, scalable. I'm sure it won't be too much of a surprise to you to realize we used Java to complete this. In actual fact, we built a complete Web-based e-mail/newsgroup system using Java Servlets, all linking back to a central e-mail repository maintained by an Oracle database. The whole thing worked like a charm.

Thank You, Oracle

At this juncture I must say a little about Oracle. Think of it as an update, if you wish. Some of you may remember one of the first articles in this series (***JDJ*** Vol. 3, Issue 8) in which I highlighted the major problems we were having with the JDBC drivers shipped with Oracle. To recap, bottom line, we couldn't keep the Oracle drivers up long enough to do anything with them. We weren't alone in this quest for longevity. Many of you recounted similar tales of woe to me after the article was published, in addition to the many I found before the article. Anyway, WebLogic came to our rescue and supplied an Oracle-JDBC driver that worked superbly.

> *"This milestone marks the beginning of a new life for us all. What am I talking about? Java 2, of course."*

One of the good things to come from the article was that I was granted an audience with Oracle at their HQ in Redwood City, California. There they expressed their surprise at my findings, but said they would send me an updated JDBC driver to test. Well, I can happily report that this new driver worked a treat. Just a shame it took the publishing of an article to get a result. But it was a result nonetheless. So Oracle, I thank you.

How Sweet It Can Be

Back to our cautionary tale. Having designed and built the system, we looked at various platforms that could handle high-volume traffic and run Java efficiently. Naturally, we looked at Sun's offerings. After much deliberation we decided to purchase an Ultra to host the system until the user base grew, then we would migrate to a bigger system. Seemed a safe bet – no point in overspecifying the hardware at this point in the proceedings. So in January 1998, and $6,000 later, we had a shiny new Sun Ultra serving our client base. It was wonderful. We were proud to be Sun users; we felt safe, in the bosom of the family, safety in numbers and all that. Bit like how the early adopters felt when they opened their Apple machines. But, as we were to discover, this "safety" came at a very high price.

What I'm about to tell you will scare you. It will have you running for Linux – or, even worse, NT. Yes, it's that bad. Whatever you do, please ask any accountants or financial controllers to leave the room. It's not pretty, what I'm about to tell you. Everything all clear?

As per the master plan, the Liquid Information service grew quickly in numbers. Only a couple of months after the release we had to look at upgrading the Sun box. Well, it did us proud all through development and beta testing. The numbers were growing rapidly and we needed an alternative. As luck would have it, Sun contacted us with a promotion they were doing. Basically, they would lend us a new E250 server for a month to see whether we would like to move up to it. So we called our local Sun reseller, MDIS, and had the machine shipped to us. It came, a huge beast of a thing, and once Oracle was installed it worked wonderfully. Ignoring the fact that we had to move the blooming thing to its own room due to the level of noise it was making, we were happy.

The Price of Safety

MDIS called us and asked whether we'd be interested in making a purchase. I indicated my overall pleasure with the machine and said, "Okay, let's talk." They came back with a price of around $6,000. Excellent, I thought. Since our Ultra was only nine months old, I said we didn't need two Sun boxes for this particular application and how much was our old box worth and would they be interested in taking it as part payment for the new box. MDIS said they would, and, after many days of evaluating the box, they came back to me with a price.

Remember, the Sun Ultra was purchased only nine months ago and, with the obligatory discount of around 40%, cost us $6,000. MDIS was now offering to take the machine from us for $1,200. What? As you can imagine, I was not a happy bunny at this stage. Within nine months the machine dropped in value by around 80%? Even a BMW won't drop that much. I was disgusted. First I thought, I know, it's the fax machine, maybe the "1" was supposed to be a "4" or even a "3". But, alas, the fax machine was not to blame. It faithfully reproduced the figures from MDIS.

Needless to say, the machine was not bought. After all, what sort of company could take a loss like that? No wonder Sun is growing at a tremendous rate. It's at the expense of us smaller companies. But do you know the ironic thing about all of this? The price of the Ultra had not moved in the six months prior to our purchasing the beast. We had been researching the price of Sun boxes for six months before making the purchase and the price had not moved. And to rub salt in the wound, in the summer of '98, seven months after buying the first Ultra, we phoned MDIS again to see the price of a new Ultra as we had two and the price quoted to us was the exact same. The price had therefore remained constant for at least nine months. But within four it had fallen 80%. I would have been really angry if we had made a decision to buy that second machine in the summer. So the E250 was quickly boxed and sent back.

After all, it was worth $6,000 the day it arrived and now that we were sending it back 30 days later it may have been worth only $5,000 or something like that. It was a risk we weren't about to take. But we still needed a server to run the Liquid Information system for our client. The silver lining in the whole episode was that we employed Java. This meant we weren't tied to any specific platform. We were free to choose. It was like hearing a choir of heavenly angels.

Relief in Sight

Looking around at alternatives we were impressed with what we saw. Our first port of call was Dell, to inquire about their end servers. Would you believe we could have bought three higher-powered machines for the price of one E250? Each machine came with far more memory, far more disk space, far more processor power…and as if this weren't enough, they were quieter.

"But they were a PC-based solution," I hear you Solaris diehards cry. Yes, you're right, but let me tell you, running Linux and our benchmark tests absolutely knocked spots off the E250. And because we had chosen the Java route we didn't even need to recompile. The hardest thing we had to do was to get the right password for FTP.

> *"No wonder Sun is growing at a tremendous rate. It's at the expense of us smaller companies."*

Now I'm sure I'm going to get tons of e-mail from you Sun representatives detailing the benefits that Solaris solutions can provide us with. But let me save you some time. Don't bother. We, the little guys, can't afford to buy into the Sun solution. The falling price of PCs that include dual processor solutions makes it cost-effective for companies to take this route. We can afford for our $1,400 solution to be worth only $200 after 12 months. This is realistic and the figures aren't scary. In actual fact, you get far more machine for your money.

For those of you who haven't discovered the wonders of Linux, you really ought to spend some time researching this piece of wonderful software. Think about it: Oracle isn't supporting it now out of the goodness of their hearts. They see the damage it can make and are sure as hell covering all bases by deploying time and effort into porting their database to it.

And that brings us nicely to Java 2, or what *was* called Java 1.2. Publishers must really love Sun for that version change, considering the number of books published bearing the 1.2 mark. Java 2 is now officially supporting Linux. Interesting, eh? Even Sun is recognizing the threat. So don't just take my word for it – have a look around, see how the big guns are treating Linux. Let us not lose sight of the fact that when you remove the need to process fancy graphical interfaces, a 300 MHz processor is a serious number cruncher.

Linux unleashes this power, which is one of the main factors for its popularity – apart from the fact that it's also free, which is never a bad thing. If that's not enough, the source code for Java 2 is now freely available, once you've signed a Web-based form. Sounds very like Linux, methinks. This in my opinion is a good thing. Control still needs to be exercised so we don't get a splintering of the Java platform, but freeing the source will allow more application areas to be explored that Java may have excluded from the shortlist.

So, the moral of the tale? Don't believe everything you read in print (except for this column, you understand – we'll draw a line here). Don't be scared to explore other options, and to breathe a huge sigh of relief that you've chosen Java as your underlying development language as this frees up your hardware decision-making process. Give yourself that pat on the back you deserve.

The Book Nook

Before I close this column down, let me recommend a must-have book. Last month I introduced this new feature and you are more than welcome to completely ignore these books. But they will change your life. This month's life changer is *Out of Control* from the keyboard of Kevin Kelly. It talks all about the biology of machines and how the computing world is striving to reproduce the well-oiled machine of Mother Nature. Kelly talks about developing perfect, error-free code, which is truly inspirational. Another thought-provoking subject area is that of evolving software, the capability for software to "evolve" a solution as opposed to being given explicit instructions to solve the problem. Every developer needs to have this book on the shelf.

On that note I shall go and extract myself from this Macintosh world, and immerse myself back within the comfort of NT.

About the Author

Alan Williamson is CEO of N-ARY Ltd., a Java consulting company with offices in Scotland and Australia that specializes solely in Java at the server side. Alan is the author of two Java Servlet books and he has contributed to the 2.1 Servlet API. He can be reached at alan@n-ary.com (www.n-ary.com) and he welcomes all suggestions and comments.

alan@n-ary.com

Programming with I/O Streams

Part 3

Learning how to write your own custom stream classes can help with special data processing requirements.

by **Anil Hemrajani**

In the previous two parts of this three-part article (***JDJ*** Vol. 3, Issue 12 and ***JDJ*** Vol. 4, Issue 1), we looked at the fundamentals of programming with Java I/O streams and the various APIs they can be used with. This month we'll conclude this article by discussing the concept of writing custom (or, *specialized*) stream classes that can process (or, *filter*) data in a special fashion.

Overview

The JDK provides several specialized stream classes, the majority of which exist in the java.io package. These classes provide a variety of functionality and usually come in pairs – that is – one for reading data and the other for writing data. For example, the java.io.FileReader and java.io.FileWriter provide the capability for file reading and writing, while the java.io.StringReader and java.io.StringWriter provide the capability to use java.lang.String as a source for reading data from and writing data to.

JDK Stream Classes

Before you decide to write your own specialized stream classes, it would be worth your while to inspect the stream classes provided in the JDK. There are two reasons for this. First, there may already be a stream class available for your purpose. Second, you can use their source and documentation as examples for writing your own.

To give you a general idea of the various streaming classes available in the JDK, Table 1 shows a complete list (excluding deprecated classes) of Stream classes available in the java.io package in JDK 1.2. Table 2 shows some additional stream classes available in the java.util.zip and java.util.jar packages.

For more information refer to the JDK 1.2 documentation at http://java.sun.com.

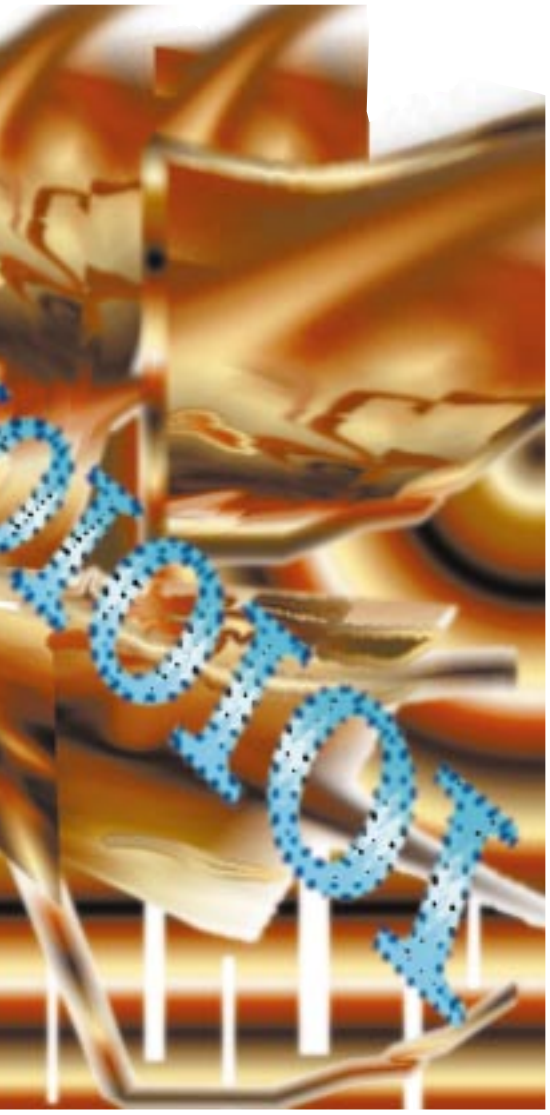
Why Write Your Own Stream Classes?

With such a variety of I/O stream classes available in the JDK, you might be wondering why anyone would ever need to write their own stream classes. Well, there are several reasons why, but it's mainly because your application might require the processing of a data stream in a special way for which the JDK has no sufficient classes.

The following are a few possible candidates for specialized stream classes.

- *Accessing Special Storage Devices* – With so many storage devices available on the market (such as tape and zip), there may be a need to access them from a Java application.

- *Spell Checker* – Java apps could use specialized stream classes that spell-check data in a stream and give back a list of misspelled words.
- *Language Converter* – A stream subclass could be written to convert words in an input or output stream from one language to another.
- *Text Search Or Screening* – Specialized stream classes could be used to search for specific words and obtain their locations in a data stream. Example uses of such classes could include a find feature



in a text editor or the ability to block out certain content (e.g. adult content) in a proxy server.

So, how does one go about writing a specialized stream class?

How To Write Specialized Stream Classes

Writing your own stream classes is relatively simple: for reading character streams, you basically extend the java.io.Reader class (or java.io.FilterReader); for writing character streams, you extend the java.io.Writer class (or java.io.FilterWriter). For byte streams, you extend the java.io.InputStream (or java.io.FilterInputStream) for reading streams; and java.io.OutputStream (or java.io.FilterOutputStream) for writing streams.

Note: To get a better understanding of the differences between byte and character streams, please refer to the first installment of this three-part article (***JDJ*** Vol. 3, Issue 12).

Filter vs Top Level Stream Classes

When developing specialized stream classes you can choose to either directly subclass the top level classes (InputStream, OutputStream, Reader, Writer) or you may subclass the Filter classes (InputFilterStream, OutputFilterStream, FilterReader, FilterWriter).

The filter stream classes seem a bit superfluous at times. However, they were originally designed for convenience because they provide default implementations for all the methods found in the top level classes; this way, you override only the methods you need to. For example, if you extend InputStream directly, you'd need to override the flush() and close() methods, even though they're not abstract methods. This is necessary because these methods have empty bodies in the InputStream class. On the other hand, when you extend the FilterInputStream class, you generally end up providing implementations for two read() methods (versus one if you extended InputStream directly).

Developing Specialized Character Stream Classes

To develop specialized character stream classes, you either subclass the Reader class for reading data or the Writer class for writing data. To develop a Reader subclass, you must override and provide implementations for the following methods:

- int read(char[] cbuf, int off, int len)
- void close()

To develop a Writer subclass you must override and provide implementations for the following methods:

- void write(char[] cbuf, int off, int len)
- void close()
- void flush()

Developing Specialized Byte Stream Classes

To develop specialized character stream classes, you either subclass the InputStream class for reading data or the OutputStream class for writing data. To develop a InputStream subclass you must over-

- BufferedInputStream
- BufferedOutputStream
- BufferedReader
- BufferedWriter
- ByteArrayInputStream
- ByteArrayOutputStream
- CharArrayReader
- CharArrayWriter
- DataInputStream
- DataOutputStream
- FileInputStream
- FileOutputStream
- FileReader
- FileWriter
- FilterInputStream
- FilterOutputStream
- FilterReader
- FilterWriter
- InputStream
- InputStreamReader
- LineNumberReader
- ObjectInputStream
- ObjectOutputStream
- OutputStream
- OutputStreamWriter
- PipedInputStream
- PipedOutputStream
- PipedReader
- PipedWriter
- PrintStream
- PrintWriter
- PushbackInputStream
- PushbackReader
- RandomAccessFile
- Reader
- SequenceInputStream
- StreamTokenizer
- StringReader
- StringWriter
- Writer

Table 1: a complete list (excluding deprecated classes) of Stream classes available in the java.io package in JDK 1.2

- CheckedInputStream
- CheckedOutputStream
- DeflaterOutputStream
- GZIPInputStream
- GZIPOutputStream
- InflaterInputStream
- ZipInputStream
- ZipOutputStream
- JarInputStream
- JarOutputStream

Table 2: Some additional stream classes available in the java.util.zip and java.util.jar packages

ride and provide implementations for the following method:

```
int read()
```

To develop a OutputStream subclass, you must override and provide implementations for the following method:

```
Void write(int b)
```

Examples Of Specialized Stream Classes Outside the JDK

Now that you have a general idea of the variety of reasons for which stream classes can be written, it would serve you well to look at some concrete examples of specialized (non-JDK) stream classes.

Simple Examples

The following two examples, TeeOutputStream and CountReader, provide simple examples of writing specialized stream classes. Later in this section, you'll see a more robust, real-world example using Divya's BackOnline commercial application.

TeeOutputStream

The class in Listing 1, TeeOutputStream, provides an example of an OutputStream subclass. The purpose of this class is not only to write the bytes to the chained output stream, but also to write them to another output stream such as the System.out device, or to a trace/debug file for debugging purposes. The functionality provided by this class is similar to the Unix tee command.

> *"Before you decide to write your own specialized stream classes, it would be worth your while to inspect the stream classes provided in the JDK."*

The code should be fairly easy to follow. Basically, this class overrides the write(int c), flush() and close() methods of its parent class, OutputStream. Additionally, this class provides a main() method for "self-testing" this class. The write() method performs the task of writing the data to two streams.

CountReader

The class in Listing 2, CountReader, provides an example of a Reader subclass. This class counts the number of characters and lines in an input stream of characters. At any point in reading the stream the get*Count() methods can be used to obtain the character, word and/or line count for the data read up to that point. The functionality provided by this class is similar to the Unix wc command.

As with the TeeOutputStream class, this class should also be easy to follow. The key method in this example is read(), which handles the task of counting characters, words and lines.

A Real World Example: Divya's 100% Pure Java Backup Application

Divya's 100% pure Java applet/application – BackOnline – is an Internet-based backup, briefcase and archival application that backs up a user's data files from the client machine to a server, as can be seen in Figure 1.

BackOnline makes extensive use of streams – everything from File Input/Output streams to Socket streams to it's own specialized Protocol and Encryption/Decryption streams. When a user backs up a file (or files) in BackOnline, the file is

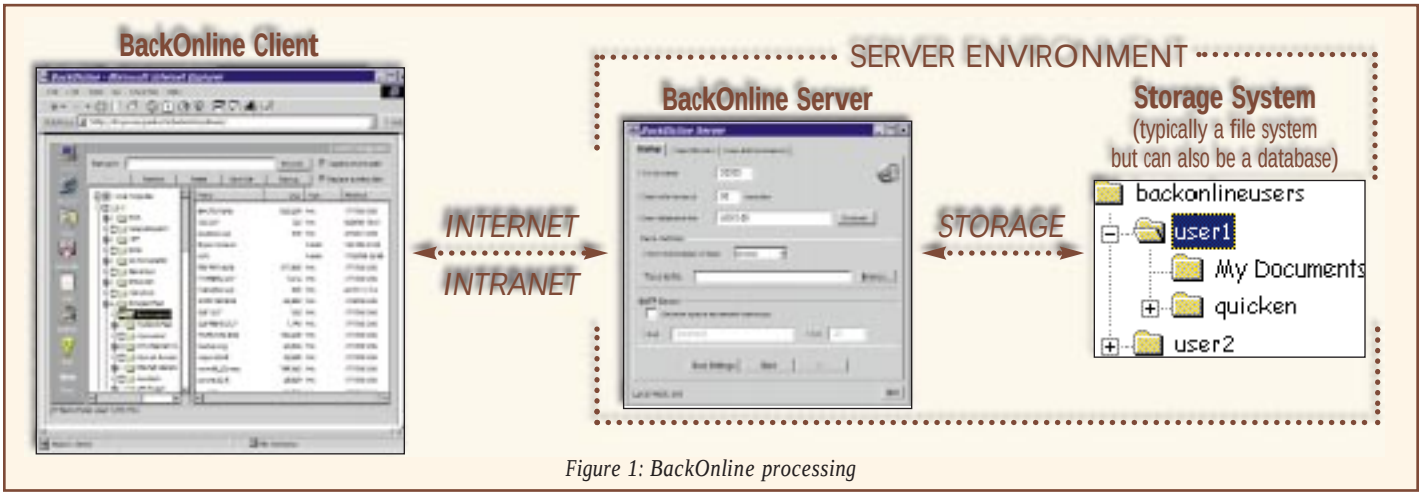


Figure 1: BackOnline processing

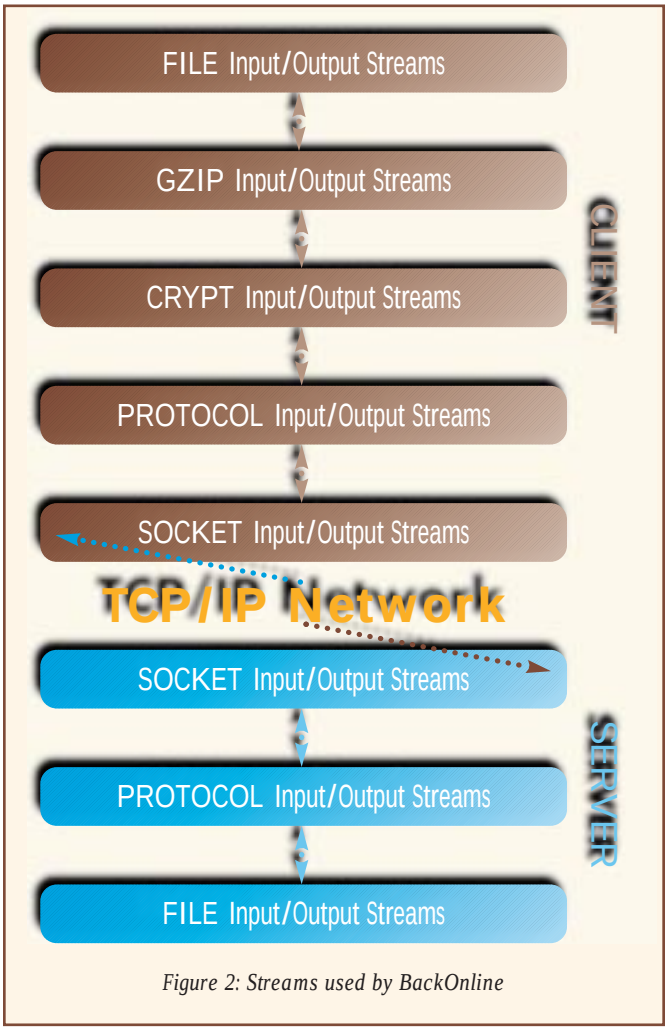


Figure 2: Streams used by BackOnline

processed via a compression stream, then an encryption stream, then sent over the wire on a socket stream. The BackOnline server receives this data using socket streams and stores it in a file using file streams. For restoring files, this process simply works in reverse. The streams in use during the backup and restore processing are shown in the Figure 2.

Though most of the stream classes used in BackOnline are available in the JDK (such as socket, file and GZIP compression), there are also two pairs of specialized stream classes written by Divya; one for the protocol used by the BackOnline client and server to communicate with each other, the other for encrypting and decrypting the stream using 56-bit DES encryption.

Protocol Stream Classes

BackOnline uses a protocol almost identical to HTTP version 1.1. Each client request (such as file backup) contains a header followed by a blank line followed by the data for that request. The stream classes automatically handle the separation of the header and data portions in the stream and provide the appropriate methods to obtain the header values and read() methods to read the actual data.

The Java source for the ProtocolInputStream class (used for reading in the protocol stream) is shown in Listing 3.

Encryption Stream Classes

When a user logs in to BackOnline, he is provided with a userid/password for authentication, as well as an optional encryption key (sort of like a second password), which is used to encrypt and decrypt the stream. In other words, BackOnline uses a key/password-based encryption/decryption scheme. Since this key is always entered by the user and is never physically stored anywhere, it provides an extremely secure mechanism for transmitting and storing the user's data.

The Java source code for the CryptOutputStream class (used for writing an encrypted stream) is provided in Listing 4. While perusing this code, pay special attention to the write, flush and close methods as they override their parent stream class' methods.

After looking at the source code for the TeeOutputStream, CountReader, ProtocolInputStream and CryptOutputStream classes, you should have a very good idea of how to write your own stream classes.

Summary

Although by now you might have an in-depth understanding of how the Java I/O stream classes work, it'll still be worth your while to peruse the JDK 1.2 source code and documentation for the various stream classes, especially the classes in the java.io package (both can be downloaded from http://java.sun.com). Remember, while I/O streaming isn't the most glamorous subject when compared to other Java technologies, it is, in a sense, the backbone of Java and Java applications since they're used in so many APIs. Hence, a good understanding of them is essential.

CODE LISTING

The complete code listing for this article can be located at **www.JavaDevelopersJournal.com**

About the Author

Anil is a senior consultant at Divya Incorporated, a consulting firm specializing in Java/Internet software solutions. Anil provides Java/Internet-based architecture, design and development solutions to Fortune 500 companies, and occasionally writes articles and speaks at conferences. He can be reached at anil@divya.com.

anil@divya.com

JdbcStore
by LPC Consulting

Java framework stores objects in JDBC databases

by **Tim Callahan**

JDJ SPECIAL: PRODUCT SHOWCASE

One of the problems involved with storing Java objects in relational databases is that the objects and their relationships don't usually map directly to relational structures. JdbcStore is a Java framework designed to help you overcome this mismatch and allow you to easily store objects in any JDBC-compatible database. Using the JdbcStore workbench, you can model your persistent objects, their relationships and how their attributes map to database tables and columns. Based on this model, JdbcStore generates Java classes for creating, storing, retrieving and manipulating your objects. An additional collection of classes for handling database connections, transaction processing, SQL queries and object caching round out the product.

How It Works

At the heart of JdbcStore is the concept of a model describing your persistent objects. Models are created and manipulated with the JdbcStore workbench and can be saved to disk. Persistent objects are called *types* within the model and their properties define how the objects are persisted along with other characteristics.

Using the workbench types can be defined from scratch, loaded from existing .class files or created from table definitions obtained directly from the database. Associated with each type are persistent attributes, attribute to database mappings, inheritance information, key attributes, object relationship definitions and cataloged SQL. Once you define the persistent object types you generate source code that implements persistence for those types. JdbcStore also generates SQL statements to create a database to store the types if needed.

JdbcStore generates a user class, a persistent extension class, a factory class and even BeanInfo classes for each type. The user class defines the persistent object and is normally only generated for types that are defined from scratch in the workbench. The persistent extension subclasses the type to implement persistence with methods such as Store(), IsDeleted() and MarkDeleted(). The factory class is used to generate object instances and includes methods such as NewInstance(), FetchAll(), FetchForKey() and FetchUsingSQL().

Within your application you use these and other JdbcStore classes to load a model, connect to the data source and work with persistent objects. The database connection is handled through a supplied ORB layer for flexibility and objects can be retrieved by using object relations or SQL queries. Related objects can be instantiated immediately or upon being referenced (lazy instantiation). Object caching and concurrency control are supported also.

Installing

I bought JdbcStore online (both Windows and Macintosh versions are available) and downloaded a self extracting archive containing the program. Running the archive starts an unpacking wizard which lets you select a location for the JdbcStore installation files. Extracting the installation files requires a product serial number and digital key, which were quickly e-mailed to me after my purchase.

You must have a Java VM installed on your computer before installing JdbcStore. Executing the JdbcStore.class file starts a simple InstallShield installation process. The only choice you have to make is the installation directory (the default is C:\LPC.) Installing JdbcStore took less than five minutes and consumed under 6 MB of disk space.

To run JdbcStore you need to add c:\lpc and c:\lpc\symantec\symbeans.jar to your CLASSPATH (assuming the default installation location.) If you have a recent version of symbeans.jar (say from Visual Café) you can use that instead. According to LPC Consulting, the JdbcStore workbench makes limited use of Symantec Beans and will be converted to JFC in the future.

JdbcStore Version 1
LPC Consulting Services, Inc.
7 Geraldine Court
North York, ON M3A 1N9 Canada
Phone: 416 510-1660
Fax: 416 446-0559
E-mail: lpcsales@ilap.com
Web: www.ilap.com/lpc
Requirements: 100% Java, works with any Java VM and JDBC driver.
Price: $99

Setting Up

I tested JdbcStore on a Pentium 133 running Windows 95, with Visual Café 2.5 and JDK 1.1.6. The database was Access 97 and Sun's JDBC-ODBC bridge was the JDBC driver. For this test I created a small Access database with two tables called FRIENDS and ADDRESSES. Using the Windows Control Panel – 32-bit ODBC Drivers – I set up this database as an ODBC data source named "testdb" using Microsoft's Access driver.

I then created a simple Person class in Java with a few public attributes that I wanted to persist in the FRIENDS table. Get and set methods were defined for each attribute destined to be persistent along with a method to print the person's name. The get and set methods are required for JdbcStore to access the persistent fields. My goal was to write an application that would do some updates on the FRIENDS database and print out each person's name and addresses.

Building a Model

The JdbcStore workbench is started by executing "java com.lpc.jdbcstore.workbench.LPCMainFrame" from the JdbcStore directory. The first steps once the work-

bench is started are to load a JDBC driver and connect to your data source. I easily connected to my test database by specifying sun.jdbc.odbc.JdbcOdbcDriver as the driver and a database URL of jdbc:odbc:testdb. Once I was connected, I quickly loaded the database definition into the workbench by retrieving the SQL catalog. Figure 1 shows the definition of the FRIENDS table that was loaded from Access.

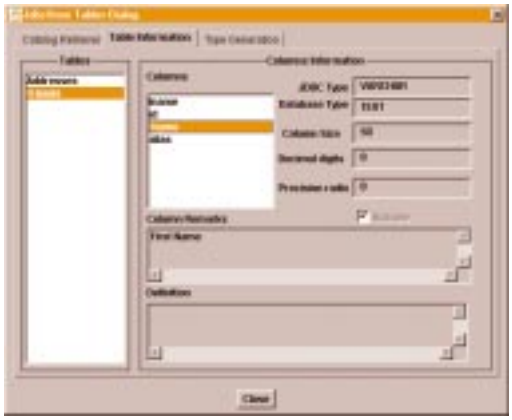


Figure 1: Table definition loaded from the database

Next I automatically generated a persistent type definition for the Addresses table. I didn't do this for the FRIENDS table since I wanted to use the Person class I'd created earlier to access that data. Using JdbcStore's class loader I added my Person class to the model and created a Person type. The public fields and methods from Person.class were visible and I added them to my new Person type.

Next, I edited my new Person and Address types to set up the database mappings. Since I had generated the Address type from the table definition, it already had attributes for each database column with JDBC mappings, as shown in Figure 2. For the Person type I had to set up the mappings to the FRIENDS table, which took me a few tries before I got the Java variable and column database types to match up properly. Once my model was complete I saved it to disk.

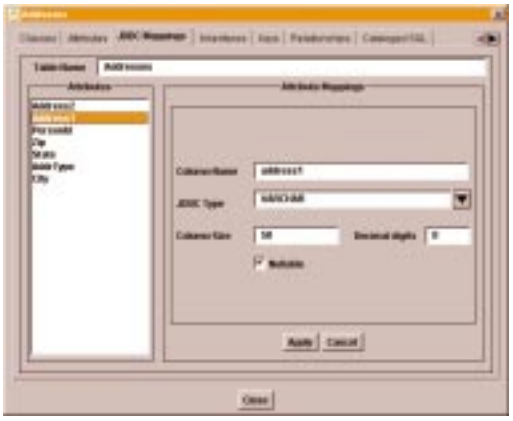


Figure 2: Attribute to database mappings

Generating Source Code

Once the model was complete I was ready to generate source code, which is done with the workbench. For the Addresses type I generated a user class (since this type was defined from scratch) along with factory and persistent extension classes. For the Person type I simply generated the factory and persistent extension classes because wanted to keep the Person.class file.

When generating source code you can choose the names and locations for the resulting Java classes. I used the default names of Address.java for the user class, lpcAddress.java for the persistent extension, and AddressFactory.java for the factory. All source code is generated from token based templates that are provided and can be modified if necessary.

Creating an Application

Writing an application (or applet) using JdbcStore is straightforward. The first things your application must do are load the JDBC driver, connect to the database and load the model. JdbcStore comes with various classes to handle these tasks. Once these steps are completed the JdbcStore classes can create, fetch, manipulate, and store objects.

A wealth of sample code and an adequate programming guide help get you started. By following the examples, I quickly created a Java application that loaded all Persons from the FRIENDS table, printed out their names, and then swapped their first name and alias values. Then I quickly extended the application to print each person's addresses under their name. To do this I sub-classed the lpcAddress class and added a PrintAddress() method instead of modifying lpcAddress directly, thus avoiding the need to reapply changes if the lpcAddress class is regenerated.

The JdbcStore run-time classes contain other useful features. The database connection is handled by an included ORB which facilitates plugging in RMI or CORBA later. The factory classes have an assortment of fetch statements for filtering and sorting objects. Classes to access the JdbcStore model at run-time are included, along with classes for managing run-time object caching and concurrency control.

To deploy an application you need to include a copy of the model, a set of JdbcStore run-time classes and any classes required by the application. To deploy an applet you need a link to your model (which can be opened using a URL) instead of a copy. The size of my model file was 6KB and the JdbcStore run-time classes took about 200 KB as .class files and 92 KB as a .jar file.

Some Caveats

JdbcStore has a few rough edges, which is expected with a product's first release. The most notable shortcomings are in the workbench's user interface. Some standard niceties such as browsing for folders in dialogs are missing. Some functions don't provide as much feedback as I would like, although details are often available in the Java console. In one instance, I created a Java compilation error in one of the generated persistent extension classes because I didn't define a key attribute for the type. This type of invalid entry should be detected by the workbench. These problems don't detract too much from the product since it is used by developers who are (usually!) adaptable to these kinds of limitations.

My other warnings have more to do with frameworks in general than JdbcStore itself. One potential problem with frameworks is that you have to adapt to their architecture and syntax. JdbcStore is simple and intuitive enough that this should not pose a problem. Another thing to keep in mind is that your applications will be tightly coupled to the framework so you should make sure it includes everything you need before you rely on it. JdbcStore seems full-featured enough to be suitable for a variety of applications. You must also be careful about modifying generated code, lest you find yourself having to laboriously reapply changes. JdbcStore is modular enough that you should be able to eliminate this problem with good design.

Summary

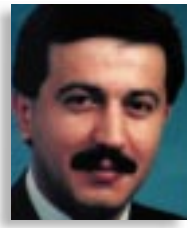
With JdbcStore you can very quickly create persistent objects and use them in applications. Some of its more powerful features include source code generation, both object oriented and SQL based data access, transaction processing, and object caching. JdbcStore's classes are intuitive, easy to use and provide enough slots and tabs to meet many needs. JDBC, an ORB layer, and customizable source code templates provide flexibility.

JdbcStore still has a few rough edges but they don't affect the product's usefulness. As with any framework you should make sure that you can use it to accomplish your specific goals. The documentation is decent and lots of sample code is included. At only $99, JdbcStore is worth checking out if you want to easily persist your objects or access existing databases with Java.

About the Author

Tim is a software developer and consultant living in Oakland, California. You can reach him through his company's Web site at www.palocolorado.com or at tcallahan@palocolorado.com.

tcallahan@palocolorado.com



ClearQuest
by Rational Software

Providing a Clear path to integrating all the platforms in your company

by **Ed Zebrowski**

JDJ SPECIAL: PRODUCT SHOWCASE

All in all, it's been a pretty good year. The hundreds of clients you've developed applications for are all happy. The software development division has grown by leaps and bounds. Then, just when it looks like you've reached the pot of gold at the end of the rainbow, trouble comes calling. It's been brought to your attention that some clients aren't getting the upgraded versions of their orders. Some complain that they're receiving versions that seemed to be fine-tuned for someone else. Still more disturbing is the fact that the documentation enclosed with the software isn't always up to date. When you investigate, you discover a whole new layer of problems. It seems that when changes are made to suit the needs of the customer, not everyone is told about it on time. The software development teams are not "talking" to each other regarding which changes are made to which applications. The technical writers aren't being kept up to speed on what documentation needs to be rewritten. The people in shipping aren't always sure which version is which, and that accounts for the embarrassing problem of wrong software being delivered to customers.

A system that enables developers to keep everyone else updated on changes made to your products is needed – something that everyone can access from his or her desktop system and that's tied into the same database.

ClearQuest from Rational Software is a Change Request Management (CRM) product designed specifically for the modern software development business. With it, a team is quickly able to track and manage changes and customizations to queries, fields and activities. It makes the process of managing change easy for everyone involved to implement, deploy and maintain.

System Requirements and Installation

I installed ClearQuest on two machines, one as a server and another on the network as user (this is the best way to test the software). Both machines were equipped with a Cyrix 150 and 64 MB of RAM. Installation went smoothly on both machines. Although the documentation I received didn't specify a minimum processor speed, it did list some other requirements:

- Windows 95 or NT 4.0 with Service Pack 3; if you're using NT, you must install ClearQuest Designer
- One of the following databases: Microsoft SQL Server 6.5 with Server Service Pack 3 or Microsoft Access (the current version of the database engine is installed with ClearQuest)
- MDAC (also supplied on the installation CD-ROM)
- About 45 MB of disk space (up to 20 MB on your system drive and 25 MB on the drive where the programs will be installed)
- An additional 15 MB of database space on the database server for each thousand records you plan to generate
- 32 MB of RAM

To install ClearQuest as the administrator, take the following steps:

1. Set up the database. I used Access, so I had to create a share for the databases. I did this by making a UNC-style path: \machine name\share name\ pathname\filename\. During installation ClearQuest creates and initializes an Access database in this directory. It's important to remember this path as it will be needed during installation.
2. Install MDAC, found in the \MDAC directory on the installation CD-ROM.

ClearQuest
Rational Software Corporation
18880 Homestead Rd.
Cupertino, CA 95014
Phone: 408 863-9900
Fax: 408 863-4120
E-mail: info@rational.com
Web: www.rational.com
Price: $1,295 ($3,295 bundled with ClearCase)

When Setup.exe is executed on the CD-ROM, you get the familiar installation wizard. You will, however, be prompted for the following information during setup:

- Server name and database name for the schema repository
- DBO login account and password for schema repository
- Read-only login account for schema repository
- Server name and database name for the SAMPL database (path if you're using Access)
- DBO login account and password for SAMPL database
- Ordinary login account and password for SAMPL database

To install ClearQuest as a user is even simpler:

1. Install MDAC from the CD-ROM.
2. Click setup.exe on the CD-ROM.

During installation it's necessary to know the schema-repository database type.

Using ClearQuest

ClearQuest is opened by selecting it from the Start menu, just as any other application in Win95. You'll be greeted by a login screen asking for the user name and password, and there's a drop-down list to select which database will be utilized.

The main window, consisting of three sections, will then appear. On the left-hand side of the screen is the workspace, which lists all available queries, charts and reports. In the top center is the Query Builder, where it's possible to create

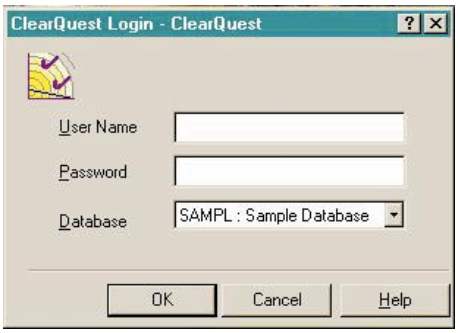


Figure 1: The standard logon box

queries and view all query results. When a record is clicked, its data is displayed in the third section – the Record Form – which covers the bottom of the window. Here it's possible to modify any query, chart or report. Once modified, the changes can be saved in a personal queries folder. New folders can be created here as well.

Suppose a customer has pointed out a slight defect in one of your products. It's now your job to make a record of the implemented changes. This process is begun by clicking "New Defect" on the tool bar at the top of the screen. It's a good idea to assign a Priority and Severity level to the change you're about to make, with No. 1 being the highest priority. It's even possible to e-mail other members of the development team automatically to inform them of the change request.

As the change to the software progresses, ClearQuest moves the change requests through various "states." When a request is first submitted, for example, it's in the "Submitted" state, where it's possible to assign the project to a specific team member. Doing so will automatically change the state to "Assigned." When the team member actually starts working on the project, the state can be changed to "Opened," indicating that someone is now hard at work on the project. This makes it effortless to see if defects have been reported and, if so, how far along you are on the road to recovery. There's even a Query Wizard that streamlines the process of creating new queries.

ClearQuest makes it possible to monitor the progress of changes in many different ways. For example, to see which developers have the most responsibility, a bar can be pulled up that graphically represents the workload for each engineer. The graph will also show defects by State and Severity. Charts may be modified by using the chart Edit Properties menu, or the Query Editor can be used to filter the records included in the chart.

ClearQuest Designer: Making Administration Tasks Simpler

The life of an administrator is never easy, so the ClearQuest Designer is formatted to streamline many tasks. Upon opening, the Open Schema dialog is displayed. It's not necessary to examine a schema unless modifications are planned. One of the many tasks the Designer can tackle is administering to users and user groups. By merely clicking the user groups icon, an administrator can quickly open dialogs to perform the following tasks:

- Define new user groups and assign users to these groups. By customizing the schema, access can be restricted to specific actions on the basis of user groups.
- Change passwords.
- Modify access and privileges. Users can be given the ability to add more users to a specific group. It's even possible to give users Schema Builder permission, and, if necessary, the user can be given the status of a Super User, which includes the ability to add and delete databases.

The task of adding and updating databases is included in the Designer as well. This is as simple as selecting the new database and supplying the requested information. ClearQuest recognizes each database by a five-character tag that is part of the ID of the records in that database. If desired, a more "user recognizable" name can be added for quick identification. Deletion of a database doesn't remove the database from the server, but merely removes it from ClearQuest's list of known databases.

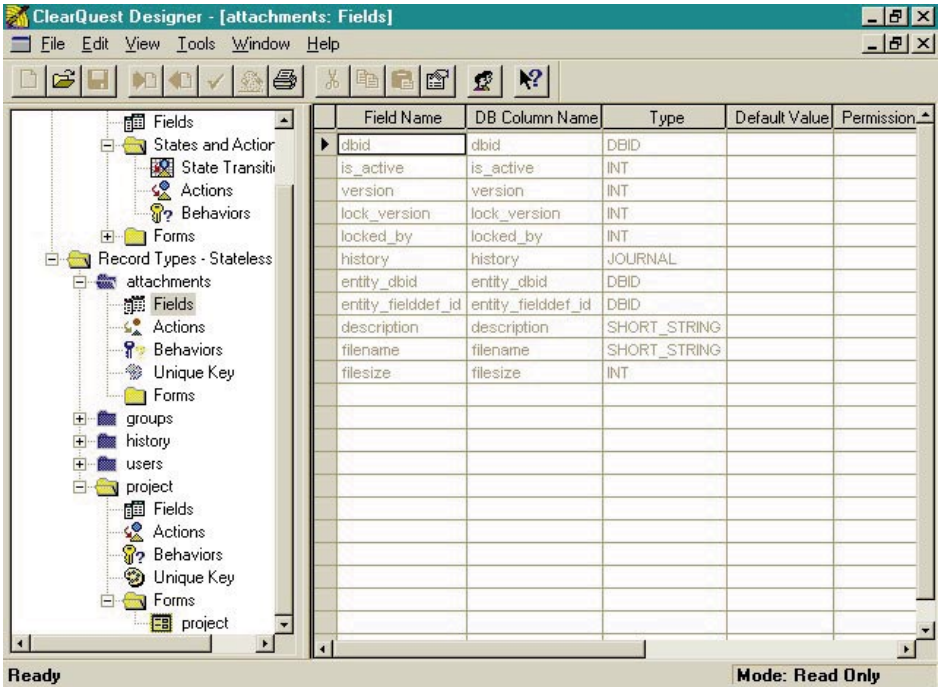


Figure 2: Looking just like the user-end application, ClearQuest Designer simplifies many administrative tasks.

Other Features

- Compatibility with Visual Basic: ClearQuest comes packaged with some Visual Basic scripts, which can be used as "hooks" to perform various tasks, and edited for more complex tasks.
- Web access: Anyone with this feature can modify change requests, generate reports or run queries.
- Integration with ClearCase SCM (Software Configuration Manager): Enables users to determine which versions of source code were modified and in what order. It's available for purchase bundled with ClearCase.

New in ClearQuest 1.1

- Stronger integration with Rational ClearCase, including support for ClearCase users on UNIX.
- New integration with Microsoft Visual SourceSafe and Developer Studio.
- New support for Oracle and SQL Anywhere databases.
- Improved Web interface adds more Windows client functionality.

After using ClearQuest, I must say that I was pretty impressed with it. If you're a single developer who services one or two clients at a time, this product is obviously not for you. For larger businesses, however, the organization and ease that the product offers makes it well worth the investment. It could be just what you need to weave your way through the chaos.

About the Author

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

zebra@rock-n-roll.com

ServletDebugger 2.0

by Live Software, Inc.

A pure Java servlet debugger and stress tester

by **Robert Tiffany**

JDJ SPECIAL: PRODUCT SHOWCASE

ServletDebugger 2.0 works with your favorite Java development tool to tackle the tough job of debugging and stress testing Servlets. This tool eliminates the usual "code, compile, cross your fingers and test" method of Servlet construction. ServletDebugger is a Java 1.1-based library that works with a simple stub class allowing you to step through your Servlet code one line at a time. Servlets can be tested with GET and POST request types as well as unlimited init parameters, header values and form data. Additionally, ServletDebugger comes complete with a built-in Web server to test and debug your Servlet using your Web browser.

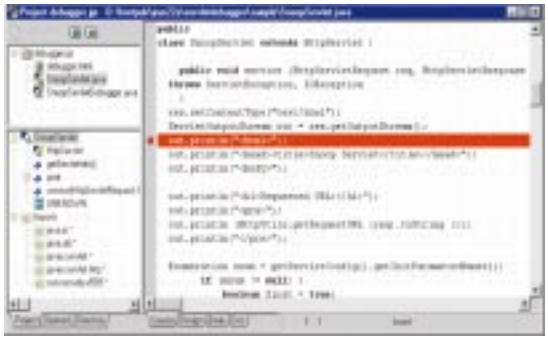


Figure 1: The JBuilder code window

Normally at this point in a software product review, I'd be giving you a long list of requirements in order to make sure this product worked with your system. Since ServletDebugger is 100% pure Java, the only requirement is a JDK 1.1 Java Development tool that runs on your particular operating system. In my case, I installed it on Windows NT Server 4.0. The InstallShield setup worked without a hitch and included the servletdebugger.jar file, a sample project and documentation in Microsoft Word, HTML, RTF and PDF formats.

The documentation gives an overview of ServletDebugger's features and capabilities, how to use it with the included sample project and a short tutorial in Servlet development. Since the sample project and associated documentation refer to Symantec's Visual Café for Java, I will describe how to set up and use ServletDebugger with Borland's JBuilder 2.

In order to debug Servlets from within JBuilder, you must first make servletdebugger.jar available to the development environment. With JBuilder open, select the "Tools" menu and click on "Default Project Properties…." Click the "Add" button on the "Paths" tab. In the "Select a Java library to add" dialog, click "New." In the "Name:" field type in "ServletDebugger." Click the "…" button to the right of the "Class Path" field. At the bottom of the "Edit LibraryClassPath" dialog, click the "Add Zip/JAR" button. Use the "Add Zip/JAR" dialog to locate "servletdebugger.jar" from your file system. Once you've found the file, click "OK" at the bottom of the "Edit LibraryClassPath." Click "OK" at the bottom of the "Select a Java library to add" dialog. Click "OK" to exit the "Default Project Properties" window.

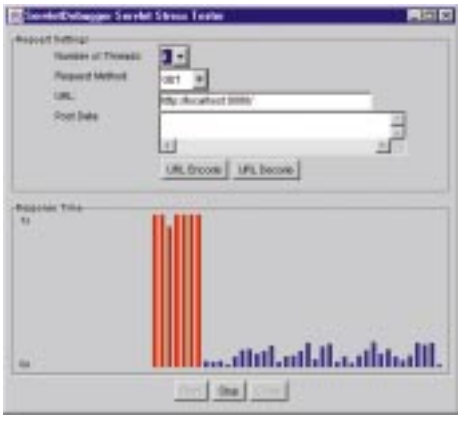


Figure 2: ServletDebugger's Stress Tester

With the environment set, create an empty project in JBuilder and then add "SnoopServlet.java" and "SnoopServletDebugger.java" from the sample Visual Café project directory. At this point you can set breakpoints in the "SnoopServlet.java" Servlet, as seen in Figure 1, and then begin debugging it by selecting the "Run" menu and clicking on "Debug." ServletDebugger gives you two ways to debug your Servlets. The normalModeTest method runs the Servlet and sends the output to the error stream System.err. The serverModeTest method uses a built-in HTTP server to allow you to submit multiple requests to your Servlet from your browser. From there you also have access to the Servlet Stress Tester shown in Figure 2 which allows you to GET and POST Form data to your Servlet for further debugging and performance analysis.

Now that Servlets have really taken off, ServletDebugger 2.0 fills a widespread debugging and performance testing need. You can now step through your code in the same way that you would if you were debugging a Java application. I think the documentation could use some beefing up in the areas of sending different request types to your Servlet using the normalModeTest method. Seasoned Java developers should have no trouble figuring things out, but a novice might. All in all, I find ServletDebugger 2.0 to be a great product that I personally find indispensable when it comes to building professional Web sites.

ServletDebugger 2.0
Live Software, Inc.
5703 Oberline Dr.
Suite 208
San Diego, CA 92121
E-mail: contact@livesoftware.com
Web: www.livesoftware.com
Price: $195
Platforms: Any platform that supports Java 1.1

About the Author

Robert Tiffany is a Senior Technology Consultant with Insource Technology in Houston, TX. He can be reached at robertt@insource.com.

robertt@insource.com

Spirit 1.0
by eVisNet

A presentation environment based on JavaBeans

by **Gabor Liptak**

JDJ SPECIAL: PRODUCT SHOWCASE

Spirit 1.0

Content development platform: Windows9x

Player platform: Netscape 4.x or IE 4.x browser

Pros:

Utilizes Java Beans (and their interactions) to provide a rich presentation environment

Presentation can connect to HTML and Java Media Framework data types

Cons:

Inconsistent UI and small UI annoyances

The product costs $299, downloadable from www.evis.net. There are no runtime fees for site deployment, but a seperate license is to be negotiated for embedding or intranet use.

The product was released in September 1998, but based on the information on the Web site, it's still undergoing development iterations.

Spirit is a presentation builder environment based on JavaBeans. It provides an extensive set of Bean-based objects (images), presentation tool transitions and effects, and also has the ability to "script" its objects using Java standard Bean interaction conventions. The product itself is partially built using Spirit, and its UI shows proof of this by providing more of a graphics workshop interface as opposed to the GUI builder interface.

InstallAnywhere Doesn't Work

Its installer utilizes InstallAnywhere, which created a shortcut to the application in the Start menu, but curiously didn't create a link to the uninstaller. The installation process installs JRE 1.1.5 but doesn't include the javac compiler required for its basic operations. Although the README states otherwise, a separate install of a JDK 1.1.x or JDK 1.1.x-compatible Java programming environment is required to make the product fully functional. When the demo version of the software doesn't have the proper key file installed, it just locks up and needs to be killed using the task manager (see Figure 1).

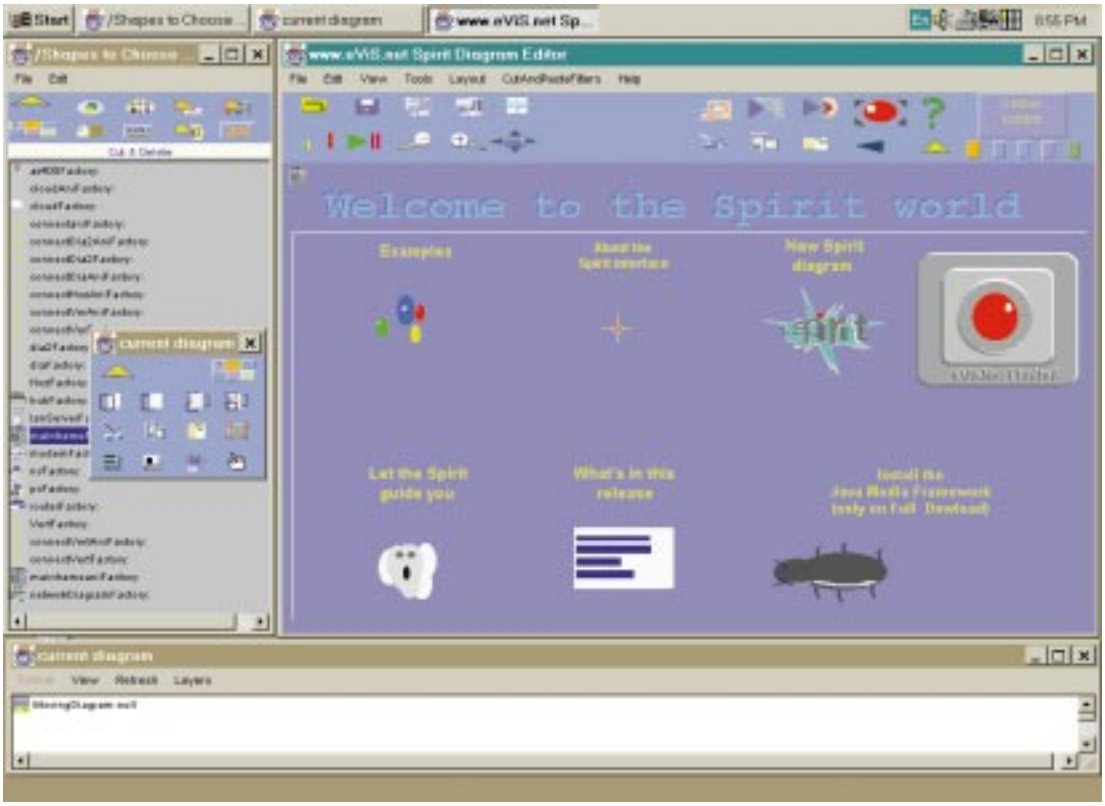


Figure 1: Builder environment

I found the product interface busy at a screen resolution of 1024 x 768. Spirit utilizes three windows and several toolbars to present its information. The right-side window is the editor, the left-side window contains a folder-based view of the available shapes (more can be imported into the system using JavaBeans) and the bottom window contains a folder-based view of the current diagrams layout. I think the product would benefit if the folder-based view could be switched to tree-based view, because different sets of controls could then be visible at the same time. The editor window also serves as

a runtime environment, presenting most of the actions on diagram without actually packaging it for the runtime environment. The current clipboard content is visible in the upper left corner of the editor window, containing the image of the shape selected in the left window to be used in the current diagram.

You are well advised to take the product demo, "Let the Spirit Guide You," offered on the Welcome screen. It provides a good presentation of the product abilities. Considering the somewhat unusual interface of the product, you may want to spend some time reading the online help as well.

When I was creating a small sample diagram to better understand the product abilities, I ran into many small annoyances. The Refresh button is named somewhat misleadingly on the Properties dialog; it seems to be for restoring settings instead. The Properties dialog doesn't provide a clear indication for some non-modifiable properties. The Esc key is used to finish resizing an object, although it's mostly used to cancel moves within similar graphics products. Spirit claims that there are two ways of reviewing events – by using the debug feature called Review Actions, and from the Edit Actions menu option – but I could find no way to review the details on existing events (except possibly by running them?).

I noticed some focusing and repainting problems inside the Builder itself, which I'm not sure whether to attribute to Java AWT quirks or to the newness of the product. The right mouse button (popup menu) usage is inconsistent, e.g. you can't use it to call up the menu in the current diagram frame. The usual helper functions of graphical editing environments (e.g. line up, same size, etc.) aren't available either. This is because Spirit has a simple grid which only snaps to position when pasting new objects or resizing them, unlike other graphical environments which will snap them all. If the product targets a generic bean builder, LayoutManager-based positioning options should be available. There seems to be no provision for listing last accessed diagrams in the file menu.

The Player's size is 170 K, which allows deploying the generated diagrams via both intranet and the Internet. Utilizing HTML and Java Media Framework data type links and the Beans based environment provides a rich presentation environment not achievable in traditional presentation tools without custom development and special add-ons.

A few other interesting features of Spirit 1.0 include:

- A geographically-based concept of a window moving over a large space containing information, thus saving screen real estate and providing some very smooth real-time zooming effects
- Spirit is an object browser and is able to load other Spirit content without the browser needing to load a new HTML page
- Tiny file sizes when deploying over the internet, as well as a zip format and a streaming option if deploying multiple presentations.

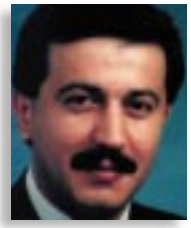
I found the WWW site busy and too colorful, concentrating on the company product(s), instead of associated information that would help site navigation. The site contains an extensive demo of Spirit which highlights the different areas in which the program is applicable.

At the time of this writing, the Player required a JDK 1.1.x-enabled browser, although I noticed some stability problems when using Netscape 4.5 on Linux. Considering Java's roots in the product, hopefully cross-platform support will soon be available.

About the Author

Gabor Liptak has been programming since 1984, and has been using high and low level object orientation design and implemantation for the past four years. E-mail him at gliptak@hotmail.com


gliptak@hotmail.com



eNetwork On-Demand Server by IBM

Big Blue ups the ante in the Java market

by **Ed Zebrowski**

JDJ SPECIAL: PRODUCT SHOWCASE

With its recent announcement of the release of eNetwork On-Demand Server, IBM has raised the stakes in the game of Java development. Now it's possible to personalize, secure, manage and deploy your Java applications from a centralized Web or Web application server. On-Demand Server enables developers and system administrators to customize Java application preferences for individuals, groups or even a specific type of client machine. All IDs, passwords, applications and profiles are stored on the server, which enables the same access policies to apply whether the client is logging on from the next cubicle or from a distant city. All files that are critical to the application are stored on the server, preventing users from corrupting or illegally accessing files and causing a system crash. This is a big advantage over the traditional client/server model.

System Requirements (Client):
- 100 MHz processor
- 32 MB RAM (end user)
- 48 MB RAM (administrators or end users requiring additional performance)
- 10 MB hard drive space
- 800x600 pixels and greater than 256 colors display

One of the following operating systems:
- Windows 95 or 98
- Windows NT 4.0 with Service Pack 3
- IBM OS/2 warp 4.0 with FixPak 4 or later

One of the following browsers:
- Netscape Navigator/Communicator 4.06 or later
- IE4.01 with service Pack 1

System Requirements (Server):
- 180 MHz processor
- 96 MB RAM (96 MB for each processor if using a symmetric multiprocessor server)
- 100 MB hard drive space for NTFS format (200 MB if FAT format)
- 800x600 pixels and greater than 256 colors display

One of the following operating systems:
- Windows NT Server 4.0 with Service Pack 3
- Windows NT Workstation 4.0 with Service Pack 3
- AIX 4.3
- PS/390 R5
- OS/2 Warp Server 4.0

Before On-Demand Server can be installed; one of the following Web servers must be installed:
- Lotus Domino Go Webserver 4.6.2 (Provided with the On-Demand Server package)
- Netscape Enterprise Server 3.51
- Netscape FastTrack Server 2.01
- Microsoft Internet Information Server 3.0 or 4.0

eNetwork On-Demand Server
Phone: 800 IBM-CALL (800 426-2255)
Fax: 800 2FAXIBM (800 232-9426)
E-mail: enetwork@us.ibm.com
Web: www.software.ibm.com/enetwork/on-demand
Price: $79 per client

Installation

Before you attempt to install eNetwork On-Demand Server, it's important to make sure your machine is properly set up to run the product. While this isn't necessary with the client-side version, the server-side of eNetwork can only function on a server-ori-

Figure 1: By using the interactive splash screen, installation will be quick and painless!

entated OS. The version I tested ran on Windows NT 4.0. It's also vital to run one of the supported Web servers that are mentioned above. The version of eNetwork I tested included the Lotus Domino GO! Web server.

When I inserted the CD-ROM into my drawer, an interactive splash screen greeted me. Among the selections I could choose from were browsing the CD-ROM, viewing the documentation, going to IBM's Web site and of course, installing the software. I installed the server on an Intel 266 with 128 MB of RAM and found it to be fast and trouble free. The workstations varied from a 133 to a 266.

If you have your autorun disabled or if you don't get the splash screen for some other reason, installation is still pretty simple. Browse the CD-ROM to Help\onDemand\en\gssetmst.htm, click on "Initial Steps," and you'll find some very good HTML documentation to walk you through the process.

Configuration Is No Harder Than Using a Web Browser

One of the features I liked best about On-Demand Server is its configuration interface. Unlike the command line interfaces of some other servers, On-Demand uses your favorite Web browser to accomplish this once-gruesome task. When launched, the Configuration Manager automatically detects any installed services and displays them in the left browser window. Clicking on any given service brings it to the right window, where a simple click or drag can add, remove or modify.

Figure 2: Your favorite Web browser is all you need to run On-Demand's Profile Manager

Not only can a browser be used for basic configuration, but to add, define and customize Java applications as well. Software changes are automatically accessed from anywhere, saving the user the aggravation of going through installations and upgrades. This profile management interface can also be used to assign user IDs and passwords, and to define software user access. I was able to configure quickly and easily from the server, the workstation and even from a remote dial-in location.

The profile manager features an extensive set of class libraries. Applets written with these class libraries can automatically retrieve the end user's data and preferences. When the user logs on, this customized set of preferences is served to the user and displayed in the Applet Launcher. These applets are cached in the browser, saving the user download time during future sessions. If a new or updated version of an applet is installed on the On-Demand Server, the cached applet is automatically updated at the next logon.

Using On-Demand Server

For testing purposes, I chose to use the demo software that came with my package. The installation, while lacking the interactive splash screen, wasn't too difficult. My demo software CD-ROM contained a directory named "onDemand-ServerWin32-Client" and a self-extracting zip file named "serverClasses.exe". It's important that these zipped files are extracted to the proper directory. Make sure it's (installed directory)\WebSphere\AppServer\web\onDemand\Classes. Next go to the directory named \WebSphere\AppServer\web\onDemand\Classes\local\nativeLauncher. Locate two files named "load1.txt" and "load2.txt". The host name in each of these files will be listed as "brqnta.raleigh.-.ibm.com". Change the name to the qualified host name of your On-Demand Server.

Here's where I committed a small error. After my installation, I couldn't get the applet launcher to behave properly. As it turned out, the support files for Win32 clients must be installed on any Win 32 machine that runs the applet launcher, including the machine hosting the On-Demand server. I assumed that the applet launcher on the Win NT server would suffice, but I was wrong. It seems that these support files contain Java native interface support for the client and are vital because the browser JVMs don't support JNI. I could have saved myself some time if I'd properly read the documentation before I began. (Dad always used to tell me to read the directions carefully before I started anything, but I would never listen!)

Once I saw the error of my ways, installation of the support files went very smoothly. This was accomplished by copying the directory named "onDemandServer-Win32Client" from the CD-ROM to "C:\onDemandServerWin32Client". In this directory there is a batch file named "startJNIServer.bat" and it's necessary to edit one line in this file, a line that reads "set PATH=WINDOWS_VERSION;%PATH%". If you're using WIN NT, edit the line to read "set PATH=winnt;%PATH%", but for WIN 95, it should read "set PATH=win95;%PATH%". After this quick edit, execute the "startJNIServer.bat" file and leave it running for the duration of the On-Demand Server operation. This will start the Java application as a server program.

With that said and done, it was time to start using On-Demand Server. The first scenario I tested involved the migration of the existing components into On-Demand Server. The first step was to migrate the existing user and group definitions. This was easily done by taking the following steps:

1. I defined a set of users and groups to the native server. On WIN NT, all I had to do is click Start – Programs – Administrative Tools – User Manager. This displays the current enterprise server configuration. I then created two groups, shipping and sales, and a user named emp1, which I made a member of the shipping group.
2. I started the On-Demand configuration console before I ran the migration tool. This made it possible to watch the entire migration process. I launched my browser to http://(hostname)/IBMWebAS/-onDemand/configure.html. It was necessary to then enter the userid and password. Agfterwards, I went to "Profile Management" in the left pane and clicked on the lists: "User Groups" and "Users". I was then able to watch these lists grow as the migration process continued.
3. I ran the "AddUsers" command from a command line. In my WIN NT example, the correct parameters were: AddUsers –source ondemandnt –destination ondemand.

The next step I took was to import all the necessary applications. Although this can be done by a manual entry of parameters, the example (thankfully) included an import file. I took the following steps:

1. I started the migration tool by using the same process as described in step 2.
2. I expanded the "Profile Management" in the left pane, then clicked on the "Software" tag to bring up the software configuration panel in the right pane.
3. I clicked on the "Import" button and turned my browser to the directory WebSphere/AppServer/web/onDemand/-Classes/local/nativeLauncher. I then selected the file "load1.txt" and clicked on "import". On-Demand Server then began to import the software definitions from this file. I was then able to view all the new applications as they were added.

> *"All in all, I found IBM's eNetwork On-Demand Server to be a pleasure to install, configure and use."*

At this point, it was time to configure the end-user desktops for employees of the shipping department. This involved the following steps:

1. I expanded the "Profile Management" in the left pane, and the "User Groups" tag. Then I clicked on the "shipping" tag and brought up the group configuration panel in the right pane. All the users in this group are displayed here. By clicking on a member, I was able to assign software permissions to that user.
2. I clicked on the "Software" tab in the right pane, which brought up the view of software assignments. A list of all available applications appeared and I was able to assign them to a user by clicking on them.
3. At this point it's possible to customize applications for the department. For example, by bringing the "Software" tabbed panel to the front of the screen, I was able to click on "PhoneDialer", and then click the "Run/Customize" button. From there, I could set up some speed dial settings for the department.

At this point, it was necessary to repeat the process for the sales department. This demo used only two departments, but On-Demand Server can accommodate as many as necessary.

If end users feel the need, they can further customize their own desktops and applications. The administrator can, of course, define sets of rules and parameters to restrict these options.

The On-Demand Server Toolkit:

Easy to use class libraries that can enhance your application's effectiveness

The On-Demand Server Toolkit consists of class descriptions, Javadoc, samples and tasks. The toolkit is automatically installed with the standard On-Demand installation. Some features included are:

License Management class library and Bean

This is a nifty feature that, among other things, has the ability to assist customers by enabling per-seat-based charging. This can inform customers when entitled licenses are soon to be exceeded. To use License Management, there must be at least one server running IBM License Use Management Runtime on the network. This can be installed on the same machine running On-Demand server or any other machine on the network. On-Demand Server will act as client to IBM License Use Management Runtime.

Log/trace facility class library

This enables centralized event logging and tracing capabilities for applets. For example, any significant change in the state of a monitored system could be logged as an event.

Profile Management class library

As described earlier, the Profile Manager can define individual users, group users and assign users and groups permission to files or to run software.

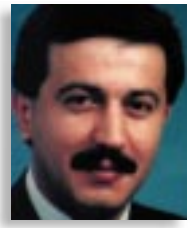
All in all, I found IBM's eNetwork On-Demand Server to be a pleasure to install, configure and use. The interfaces are user-friendly, the Java classes are state of the art and through the whole test, I did not experience a single crash.

If you're a Java developer looking for a more efficient way to manage and deploy your network-based applications or if you're a network administrator who needs a reliable and trustworthy network, place all your chips on eNetwork Server On-Demand. You can't lose!

About the Author

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

zebra@rock-n-roll.com



StudioJ
by Rogue Wave Software

Create that GUI quickly and easily

by **Ed Zebrowski**

StudioJ
5500 Flatiron Pkwy
Boulder, CO 80301
Phone: 888 442-9641
Fax: 303 447-2568
E-mail: sales@roguewave.com
Web: www.roguewave.com
Price: $995

JDJ SPECIAL: PRODUCT SHOWCASE

So you've got a nice Java application in the works. Your team has toiled for weeks to get it just right. Finally, the magic moment arrives. The development team has a meeting, and every part of the code is humming perfectly, doing just what you intended it to do. Now comes that final step, when the code is brought together into one big application. There's just one last detail – it's time to make the program "user friendly," which usually means, among other things, adding a GUI (graphical user interface) so the end users don't have to be programmers or rocket scientists to use it.

Although there are many products on the market that can do this, you, like your end users, want something powerful, yet easy to use. After all, you just spent all that time and effort climbing up the development mountain, so why not take an easy road down?

If this sounds like your situation, I recommend you take a look at StudioJ by Rogue Wave Software. StudioJ is a very fine set of development tools that will add the needed interfaces to your application with a minimum of cost and effort.

System Requirements and Installation

StudioJ will run on any JDK 1.1x-enabled system. (It was recently announced that Rogue Wave products with JDK 1.2 support will be available in the first quarter of 1999.) If you want to use StudioJ with JavaBeans, you must also use an Integrated Development Environment (IDE) that supports JavaBeans.

I installed StudioJ over the JDK 1.1.7 by running the class file named "setup.class" from the CD-ROM. An installation program initiated, and setup was a no-brainer from there.

After installation, I soon discovered that StudioJ isn't one, but four powerful development tools packaged together in one suite. These components are available separately, as well. Let's take a quick look at each of these libraries and how the developer ca benefit from them.

Quickly Create Buttons, Bars and Panels with Blend.J

There's no doubt about it, end users are unlike developers in one important aspect: they don't like to type at command prompts! Any application that attempts this is doomed to failure, therefore, it's necessary to add GUI components that are familiar and easy to use. The most common of these are buttons and "click here" bars. Blend.J contains classes for over thirty different widgets, starting with the familiar bars and radio buttons, and ranging to some pretty tricked-out interface components. Some of them include:

- ToolBar – Provides your users with an easy shortcut to the most commonly used menu items. Images that were created in other development environments can be reused by incorporating them into the toolbar. The ToolBar class is a simple java.awt.Panel with added border functionality. The image objects can be displayed with the use of any layout manager.
- ComboBox – A one-line editor with an associated drop-down panel that enables the user to click on the item of choice. This also includes a calendar and a calculator.
- GroupBox – Groups components together in a field with a decorative border. With groupbox, it's possible to reposition the text to any part of the field.
- DateTimeField – Enables the use of a local-aware time or date field. This can be configured in any practical way.
- CellGrid – A quick way to create spreadsheet-like tables to display and manage data.
- Animation – Enables the easy creation and display of a looped set of images at a specified rate.
- ImageButto – A set of classes that lets you create buttons that display graphic images instead of ordinary text. When used with the Animation class, you can display an animation loop on the button rather than a static image.
- ProgressBar – Shows the progress of lengthy operations in bar graph format. This is most useful for use with printing or downloading. At a glance, the user can tell how much longer the operation will take, or if the system has locked up.

Figure 1: Simple buttons and bars like these can be easily added with Blend.J

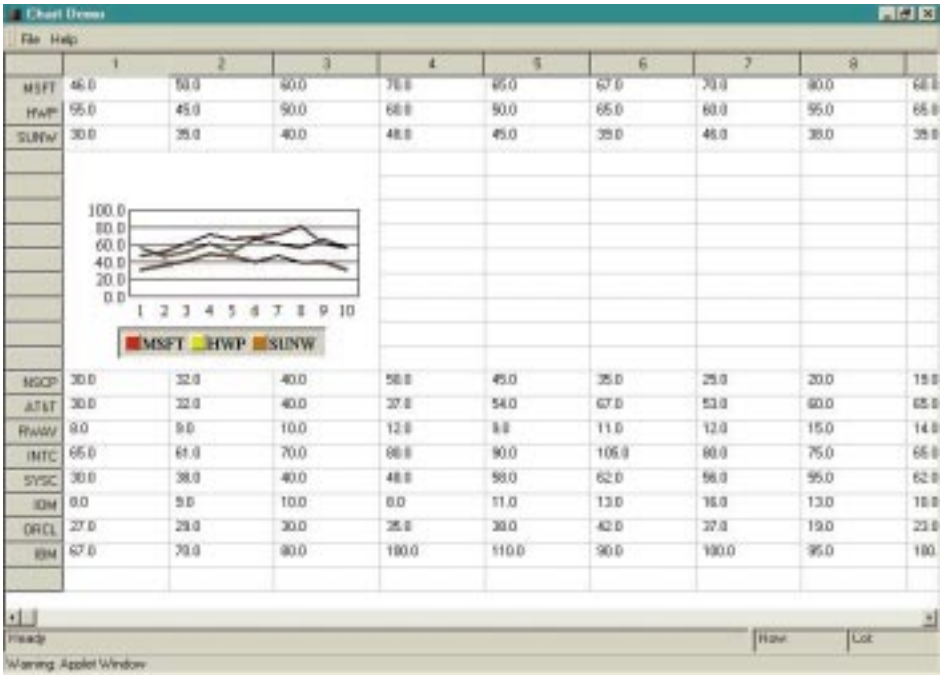


Figure 2: With Chart.J, complicated charts are no longer a big task.

Emphasize the Meaning of Data with Chart.J

Whether it's a comparative study of touchdowns scored by different teams, or how the annual departmental budget will be spent, nothing seems to drive home the meaning of data like good charts. Chart.J includes a set of Java packages and Beans that enable you to use dynamic and customizable charts in your application. You may choose from built-in chart types or a set of charting primitives that enable the creation of your own specialized chart types. Here are a few of the listed features of Chart.J:

- *JavaBeans version of all charts* – Chart.J features JavaBeans components for each chart type and many chart parts. Chart.J can be used in a Beans builder, and therefore charts can be included in the application without writing code.
- *Easy to use* – When not using a Beans builder, the set methods provided in the Chart class and Style Customizers give total control of the appearance of charts with only a few lines of code.
- *A wide range of standard chart types* – These include line charts, multi-row, multi-column and stacked bar charts, pie charts, multi-row and stacked area charts. These can be displayed in either two or three-dimensional versions.
- *Chart overlays* – Sometimes it's necessary to combine two or more charts to show how different sets of values relate to each other. Chart.J includes two prebuilt overlay charts to suit this purpose.
- *Fast and easy customization of appearance* – Chart.J supports a wide range of properties that control a chart's visual properties. These can be set in a Beans builder or through set methods.

Interface with Databases by Using DBTools.J

One of the most challenging aspects of programming is creating an application that must interface with a database. The long hours of fist pounding and head scratching can now be eliminated by the use of DBTools.J, a high-level Java API for interfacing with relational databases. DBTools.J doesn't use SQL statements, but classes that encapsulate database elements such as tables and stored procedures, or operations such as select and insert. Some of the features that make your database interfacing tasks easier are:

- *Connection pool management* – Rather than manipulate connections, you can create pools of connections, get these connections from the pool and return them when you're finished. This can greatly simplify the connection management in a multithreaded application.
- *Error checking control* – Various types of runtime error detection can be turned on and off. For example, during development, checking can be left on for debugging purposes. Once finished, this can be turned off to optimize performance.
- *Thread safety* – Shared classes are multithread safe.
- *Extensibility* – All core abstractions can be extended through inheritance. This allows customization to meet future application requirements.
- *Portable SQL* – High-level abstractions are transparently converted to the SQL dialect used by the server in which they are executed.
- *True Cursor objects* – The DBTools.J Cursor class takes on the functionality of a cursor, thus hiding the underlying implementation details.

Bring Those Dull Grids to Life Using Grid.J

The use of grids is vital in presenting data to end-users. A colorful, animated or interactive grid will not only emphasize the data presented, but will enhance your end-user's experience as well. Grid.J provides an easy way to create such grids. This includes full formula support to turn a grid into a fully functioning spreadsheet, utilizing automatic reference updates and circular dependency checks. Some of the more noteworthy features include:

- Java Beans support
- Printing support
- JDBC support for database connectivity
- Formula support
- Grid can be made to act as a tree
- Support for the merging of cells
- Transparent background supported
- The browsing of external data with one virtual override
- Smooth tracking of row and column size
- A wide variety of controls for use as cells in the grid
- Row and column headers can act as pushbuttons
- The embedding of custom controls
- Undo and redo support for grid operations
- Object-oriented architecture

One such use for Grid.J that comes to mind would be the creation of an interactive spreadsheet that can be threaded through to DBTools.J for a user-friendly database interface.

Perhaps you need a few simple navigational buttons to guide your end users through an application. Maybe you just want to use some interactive charts to demonstrate next year's projections. You may be unlucky enough to have been charged with the task of setting up a very complicated database interface, using spreadsheet-type input. No matter what your GUI needs, StudioJ will be right for you. I found it to be easy to install, easy to use and best of all, you don't need a supercharged system to run it. Pound for pound, buck for buck, this set of powerful development tools is hard to beat.

About the Author

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


zebra@rock-n-roll.com

Novera jBusiness4

by Novera Software

Just the thing for all your mission-critical distributed application needs

by **Jim Milbery**

JDJ SPECIAL: PRODUCT SHOWCASE

Novera Software announced the availability of the most recent release of their application server software at the Java Business Expo in New York in the early part of December. Version 4 of Novera Software's jBusiness application server features a new Component Constructor utility along with a new Management Console and support for Enterprise JavaBeans. The result is a powerful combination that's worth investigating if you are looking to deploy applications using an application server.

Installation and Configuration

I installed the software from CD-ROM, but you can also take a tour of the software using the "jBusiness Experience" link from Novera's Web site. The installation is packaged using InstallShield and I was able to get the software installed and running within minutes. jBusiness is tightly integrated with LDAP, and previous releases of the software required you to have directory services already installed in order to work with the product. To simplify the process of testing the software, Novera offers two installation paths for this new release. If you wish to install the software on a standalone development and deployment machine you can use the Evaluation Installation. Should you wish to work with an LDAP server you can choose the Custom Installation path, which allows you to either configure jBusiness to work with your pre-existing LDAP server or to install the bundled UMICH directory server. The evaluation installation is completely automated, but you'll need to gather some information about your existing LDAP installation if you want to make use of directory services. In both cases you'll need to have a copy of JDK 1.1.6 installed to work with the Business Object Constructor or the Component Constructor.

Novera ships the JRE 1.1.6 run-time environment for the server, but you'll need to download the JDK to work with the tools.

Building an Application

The application server marketplace is getting to be a crowded arena, but Novera is differentiating itself by focusing on their Distributed Business Objects. The jBusiness application server is completely written in Java, and Novera has tested the server on a variety of server platforms including Solaris, HP-UX and Windows NT. Novera's Business Objects are Java classes that have been mapped to relational database objects. Business Objects are further subdivided into two parts, the Business Object Constructor, which is used to create the code and the Business Object Container that performs the database access on behalf of the applications.

The Business Object Constructor interfaces with the relational database, and the first task for any development effort is to connect the constructor interface to your relational database. I was quickly able to connect jBusiness to my existing Oracle database for a fictional university using the latest JDBC thin-client Oracle JDBC driver as shown in Figure 1.

The Business Object Constructor connects to your relational database and allows you to construct Java classes that

Novera jBusiness4
Novera Software, Inc.
Burlington Office Centre
25 Corporate Drive
Burlington, MA 01803
Phone: 888 NOVERA1
Web: www.novera.com
Price: $3,495 per Developer, $9,995 per server / $350 per concurrent user for Deployment

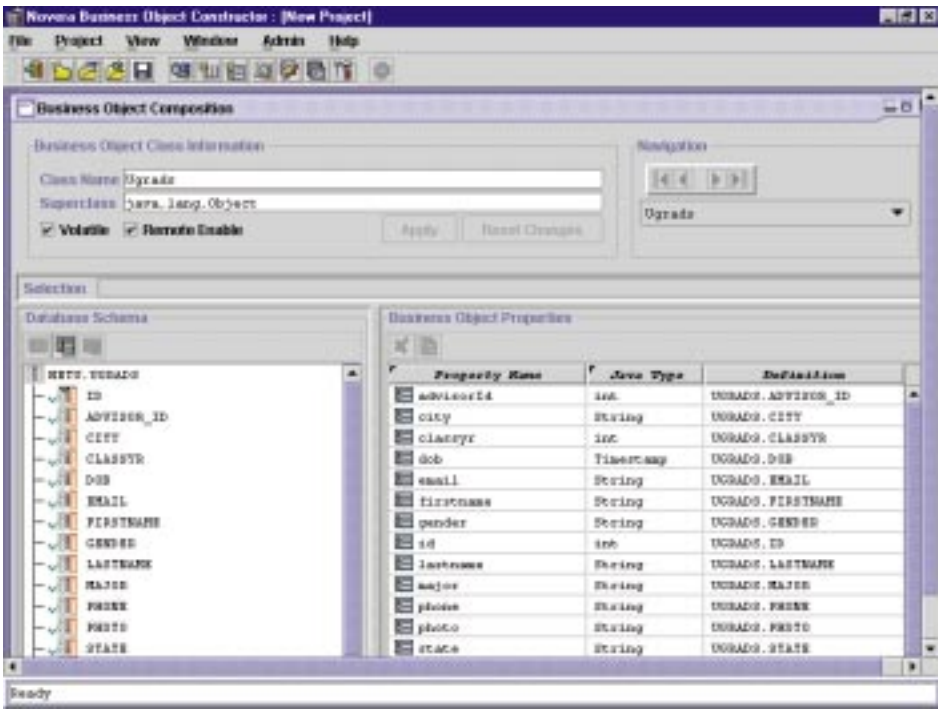


Figure 1: The Business Object Constructor

represent your relational database objects. These objects manage all of the SQL tasks such as generating queries, executing queries and converting query results into Java objects. With this release of jBusiness you can now build a single object that combines information from multiple tables, and a single relational table can be deployed in multiple business objects. This makes it much easier to create custom business objects that accurately represent the manner in which your applications use the underlying data.

The development interface automatically picks up the tables and key definitions for you, but I wasn't able to view database procedures through the mapping facility. Although there is ample documentation for both HTML and PDF format, none of the development and management consoles provide any online help. Developers that are used to having context-sensitive help within the development environment will find this frustrating. Strictly speaking, the Novera product is not a Java application development environment in the same vein as Oracle JDeveloper, Symantec Visual Café or Borland's JBuilder. The Business Object Constructor wraps complex database objects as a series of Java classes, which you can then edit or compile using your favorite Java IDE.

The constructor itself is written in Java and if you are used to working with an IDE that is written in C/C++ you will find that the constructor can be a little slow, although I was able to run both development and deployment on a single machine without any problems. I quickly built several business objects using my sample database, and Novera provides a tutorial guide to help you get up to speed with creating and deploying business objects. This can be a trifle confusing at first and I would recommend that you follow the tutorial the first time through to see how the whole process works. In general, I considered the development environment easy enough to work with for a savvy Java developer, but novices will have a more difficult time getting up to speed.

During the process of creating and deploying your business objects, you're provided with the ability to configure the caching settings for the object. JBusiness offers a sophisticated set of caching options and you can define a particular business object as being either volatile or nonvolatile. The default setting for an object is volatile, which causes the server to read the item from the database automatically when it's referenced within a transaction. Objects that change frequently, such as inventory levels, are best implemented as volatile objects. On the other side of the equation, business objects that are static in nature such as tables of state codes or part numbers can be delivered as nonvolatile objects, in which case they will be read from the cache.

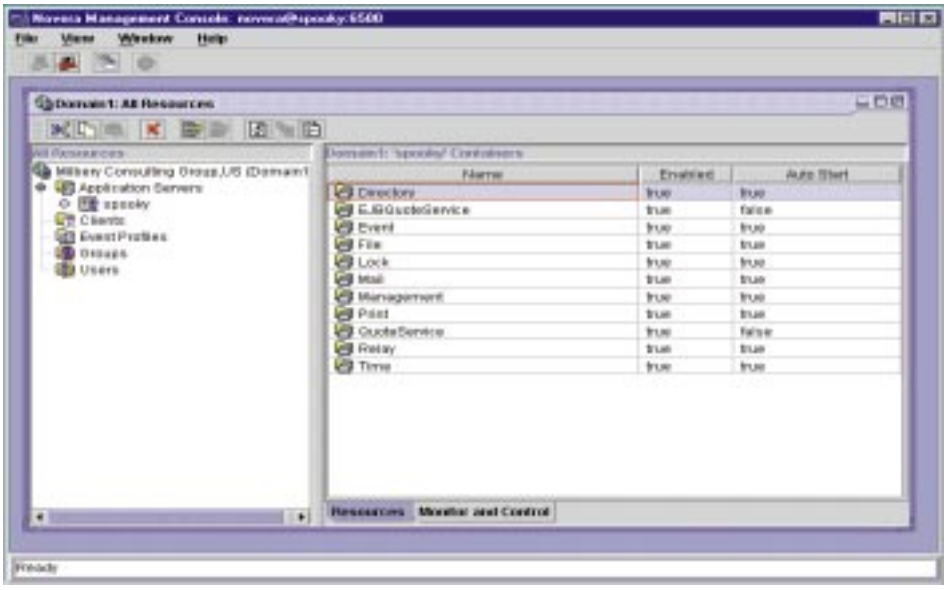


Figure 2: The New Management Console

Novera even offers a complete query language for Business Object Containers that allows you to have an incredible level of control over the caching machinations of your objects. The installation CD comes equipped with a number of sample applications, and you can use these as a road map for building and deploying your own applications.

Deploying and Managing Applications

While Novera supports the complete gamut of Web applications, including servlets, it's been designed with object management in mind. The New Management Console (see Figure 2) offers a complete environment for managing your objects and the various services that Novera provides. The emphasis within the management console is keeping track of the various objects that you've put into production. Novera supports the use of multiple application servers and you can easily display configuration information and performance for any of the services that are deployed to the server. The management console itself offers a mix of monitoring tools including charts and graphs, as well as a number of wizards for configuring the application server.

Once I had created my University Business Objects and compiled them, I was able to use a wizard interface within the management console to deploy these objects to the server. Novera is designed as a Java application server and you can connect your jBusiness objects to your favorite Web server. The management interface supports a multidocument style of display so you can monitor multiple windows at the same time, which is a nice feature. Furthermore, Novera offers an extensive API that you can use to customize the entire management environment. The jBusiness server supports a number of fault-tolerance features such as load balancing, management server failure detection and object method-level failover. I would recommend that you leaf through the extensive System Administrator's Guide before you begin designing your applications so you can get an overview of the complete capabilities of the jBusiness server.

Summary

Novera has made numerous improvements on the latest version of their software, and existing users will be pleased with the additions and changes. The jBusiness product is positioned from a pricing and features standpoint to compete in the Enterprise Application Server space. I would recommend that you evaluate jBusiness if you are looking to build large-scale Java-based server applications. JBusiness is aimed at the seasoned Java developer who is familiar with using and deploying mission-critical distributed applications.

Test Environment

Client/Server: Dell Pentium II 200 MHz, 64MB RAM, 4 Gigabyte disk drive, Windows NT 4.0 (Service Pack 4), ViewSonic 15-inch SVGA monitor, 3COM Etherlink XL Adapter and 8X CD-ROM.

About the Author

Jim Milbery is an independent software consultant based in Easton, Pennsylvania. He has over 15 years of experience in application development and relational databases. Jim can be reached at jmilbery@milbery.com, or via his Web site at www.milbery.com.z

jmilbery@milbery.com

Programming Correctness, Assertions and Exception Handling

Classes in an OOP language should be known by the methods they provide – and by the behavior and formal properties of those methods

by **Qusay H. Mahmoud**

Program correctness may be viewed as proof that the computation, given correct input, terminated with correct output. The person who invokes the computation has the responsibility of providing the correct input, which is a precondition. If the computation is successful, we say that the computation has satisfied the postcondition. The Eiffel programming language (www.eiffel.com) for example, encourages programmers to provide a fully formal proof of correctness by writing assertions that may appear in the following roles:

- *Precondition* – a condition an operation's caller agrees to satisfy. In Eiffel the keyword require introduces a precondition.
- *Postcondition* – a condition that the method itself promises to achieve. In Eiffel the keyword ensure introduces a postcondition.
- *Invariant* – a condition that a class must satisfy at all times. The invariant keyword introduces an invariant in Eiffel.

These three roles collectively support what is called the *contract model of programming*, a model that's well supported by the Eiffel programming language. Java, on the other hand, doesn't have built-in support for this model. Actually, Java doesn't even have built-in support for assertions. However, Java has a far superior feature to assertions – exception handling (see the Java tutorial on JavaSoft, "Handling Errors with Exceptions," for details).

C-Style Assertions

An assertion is a Boolean expression that, if evaluated as false, indicates a bug in the code. This mechanism provides a way to detect when a program starts falling into an inconsistent state. Assertions are also excellent for documenting assumptions and invariants about a class. Using assertions helps programmers write better code in terms of correctness, readability and maintainability. Thus they improve the odds that the behavior of a class matches the expectations of its clients.

In C/C++ you can use assertions through assert. In ANSI C, assert has the following definition:

```
void assert(int expression)
```

The assertion will be executed only when the macro NDEBUG is undefined. The program will be aborted if the expression evaluates to false (0); if the expression evaluates to true (non-0), assert has no effect. For performance reasons, however, you should write assertions into software in a form that can be optionally compiled. Thus assertions should be executed with the code only when you're in the debugging stage of software development -- that's where assertions will really help in flushing out errors. *Note:* Assertions represent a uniform methodology that replaces the use of ad hoc conditional tests.

Implementing Assertions in Java

You can implement assertions in Java quite nicely. First, create a new exceptions class for your assertions class, as shown in Listing 1.

Listing 1 extends RuntimeException, a direct subclass of Exception. Basically, there are two kinds of exceptions: checked and unchecked. The possible exceptions in Java are organized in a hierarchy of classes rooted at class Throwable, which is a direct subclass of Object. The classes Exception and Error are direct subclasses of Throwable. Instances (and subclasses) of Error and RuntimeException are called *unchecked exception(s) (classes)*.

In checked exceptions the Java language

Listing 1: An Exceptions Class

```
/**
 *AssertionException.java
 * @author Qusay H. Mahmoud, dejavu@acm.org
*/

AssertionException.java public class AssertionException extends
RuntimeException {     public AssertionException() {
super();      };     public AssertionException(String msg) {
super(msg);     } }
```

Listing 2: An Assert Method

```
/**
 *Assert.java
 * @author Qusay H. Mahmoud, dejavu@acm.org
*/

Assert.java public class Assert {     public static boolean NDE-
BUG = true;     public static boolean assert(        boolean
expression)              throws AssertionException {          if
(NDEBUG && !expression) {            throw new AssertionExcep-
tion();          }          return true;
    } }
```

Listing 3: Implementing the Assert Class

```
/**
 *Test Assertion.java
  * @author Qusay H. Mahmoud, dejavu@acm.org
*/

import java.io.*; public class TestAssertion {     public static
void main(String argv[]) {          BufferedReader is = new
BufferedReader(new InputStreamReader(System.in));
System.out.print("Please input a number: ");
        System.out.flush();          String ans=null;
        try {            ans = is.readLine();          }
catch(IOException e1) {            System.out.println("Error:
"+e1);          }          int n = Integer.parseInt(ans);
// Assert.NDEBUG = false;          // Uncomment the above line
// to turn assertions off.
        Assert.assert(n > 0);          // Use n.       }
}
```

checks at compile time whether a Java program contains handlers for exceptions. However, unchecked exceptions (as in RuntimeExceptions) are excluded from this checking. Thus Java simplifies the programmer's task by not requiring you to declare such exceptions, which could be an irritating process.

The code in Listing 2 implements an assert method, which takes a Boolean expression as an argument and checks whether it's true or false.

If it's false, your new exception, AssertionException (declared in Listing 1), will be thrown; otherwise nothing happens. Note that in the assert method you're also checking whether the value of NDEBUG (which is Boolean) is on or off. If it's on (set to true), then the assertion is to be executed; otherwise it should have no effect. Also note that clients who use the Assert class should be able to change the value of NDEBUG from their own programs.

Listing 3 provides a simple demonstration of how to use the Assert class. If you don't want assertions to be executed as part of the code, you could declare the line:

```
Assert.NDEBUG = false
```

which is really an easy way of turning off the execution of assertions.

If you run the program in Listing 3, you'll see the following output:

```
% java TestAssertion
Please input a number: -1
AssertionException
      at Assert.assert(Assert.java:5)
      at TestAssertion.main
        (TestAssertion.java:15)
```

A word of caution: never use the Assert class expression that involves side effects. Writing

```
Assert.assert(++i > 0)
```

for example, isn't a good idea, because the variable i won't be incremented if Assert.NDEBUG is set to false!

The Downside of Assertions

The use of assertions replaces the ad hoc use of conditional tests with a uniform methodology. This methodology doesn't allow for a repair strategy to continue program execution, however. This means that when an exception is detected, the program aborts with no recovery mechanism. This is certainly preferable to undefined behavior when something goes wrong, but it's unacceptable for a wide variety of applications. For example, a program that acquires resources (e.g., a lock on a database) shouldn't be allowed to abort without releasing those resources. A superior feature to assertions is built into Java – exception handling. It's superior in the sense that it allows you, using try/catch/finally, to throw an exception object instead of aborting.

Conclusion

When writing programs, you'll find it's a good idea to put checks at strategic places for violations of basic assumptions. These checks help in debugging code. The Assert class implemented in this article provides a convenient way for programmers to abort Java programs while printing a message stating where in the program the error was detected. Remember, these messages are for us – the programmers – not for users. Exception handling in Java is a superior methodology for handling errors when something goes wrong.

About the Author

Qusay works for Etisalat College of Engineering, UAE. Previously he worked for Newbridge Networks and Carleton University, both in Canada. Qusay is the author of an upcoming book on distributed programming with Java. You can reach him at dejavu@acm.org.

dejavu@acm.org

FIRST LOOK

Oracle Extends Support for Standards Based SQLJ

SQLJ brings Java closer to home

by **Oracle's SQLJ Development Team**

Oracle's development team is excited about the news that SQLJ has achieved standard status (ANSI X3.135.10-1998).

"It will undoubtedly be a catalyst in the adoption of Java by enterprise application developers. SQLJ not only allows SQL to be incorporated into Java programs in a standard way, it paves the way for database and Java tools vendors to bring Java closer to the enterprise environment. Oracle has been a firm supporter of SQLJ right from the early days when it cofounded the SQLJ consortium which submitted the SQLJ proposal to ANSI."

This *Java Developer's Journal* feature article will introduce you to the nuts and bolts of SQLJ with a look at Oracle's implementation and full support for this technology.

What Is SQLJ?

SQLJ provides a standard to embed SQL statements in Java programs. When writing an SQLJ application, a user writes a Java program and embeds SQL statements in it, following certain standard syntactic rules that govern how they can be embedded in Java programs. The user then runs an SQLJ translator, which converts this SQLJ program to a standard Java program and replaces the embedded SQL statements with calls to the SQLJ runtime. The generated Java program is compiled, using any Java compiler, and run against a database. The SQLJ runtime environment consists of a thin SQLJ runtime library that's implemented in pure Java, and in turn calls a JDBC driver targeting the appropriate database.

How Does SQLJ Work?

An SQLJ program is typically compiled in two steps. In the first step, an SQLJ Translator translates the SQLJ application; in the second step, any Java compiler can be used to compile those Java files. Figure 1 shows the steps involved in writing SQLJ applications.

SQLJ Translator

An SQLJ Translator performs two important functions:

- *Translating the SQLJ Source Code* – The SQLJ Translator translates the SQLJ application into a Java application with calls to the SQLJ runtime; it replaces the SQLJ clauses and generates a set of standard Java source files.
- *Type Checking* – SQLJ Translator performs SQL syntax-checking, schema-checking and type-checking of host variables at translation time if the logon information to the database is provided. This is performed statically, i.e., all SQL statements in the program are checked, irrespective of the actual code paths executed at application runtime.

Compiling and Running an SQLJ Application

A compiled SQLJ application is a standard Java program that can run wherever a Java VM, the SQLJ runtime library and a JDBC driver are available. There are three important aspects to consider regarding executing SQLJ applications:

- *SQLJ Runtime* – At runtime, an SQLJ application communicates with a database through the SQLJ runtime library, which is a thin layer of pure Java code above a JDBC driver.

Figure 2: SQLJ runtime configurations

- *Type Safety* – SQLJ associates the properties of result sets and database connections with the types of Java objects that represent them so those types can appear in the interfaces between separately developed Java components. For example, the shape of rows of an iterator object (the number and types of the columns) is encoded by its type (its class). That iterator class can appear as the types of parameters and results in the inter-

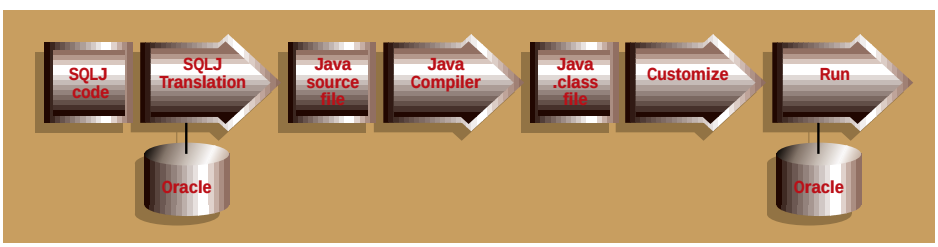


Figure 1: SQLJ development process

faces between software components. Therefore, components can exchange SQL result-set data as type-safe, first-class objects with known row shapes, which allows the SQLJ Translator and Java compiler to check accesses of column data.

- *Binary Portability* – Applications translated by SQLJ can access any database with an implementation of JDBC or a compliant implementation of the SQLJ runtime API. This property of binary portability allows compiled applications to be portable not only across platforms, but also across different vendors' databases.

SQLJ Deployment

From a platform point of view, the only requirements for running an SQLJ program are the availability of:

- The SQLJ runtime library
- A JDBC driver; Oracle's SQLJ Translator can be used with any JDBC driver
- A Java Virtual Machine, where SQLJ programs will execute

Deployment Scenarios

SQLJ programs can be deployed in a number of different configurations, including:

- Fat or thin clients
- Middle-tier Java Web servers or application servers
- A stored program on the Java Virtual Machine integrated with the Oracle8*i* database

Since the SQLJ runtime library is a thin layer of pure Java code that sits above the chosen JDBC driver, users must choose the JDBC driver best suited for their particular deployment configuration. Figures 3-6 illustrate how Oracle's SQLJ Translator can be used in combination with Oracle's own JDBC drives in four different deployment configurations.

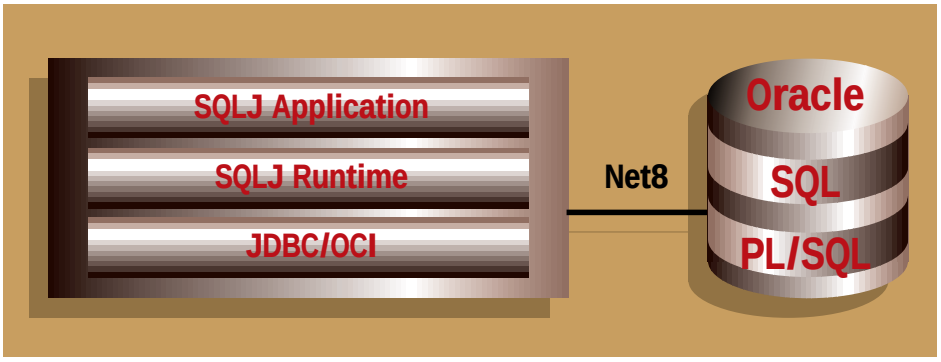


Figure 3: Using SQLJ in a two-tier server configuration uses the Oracle OCI driver.

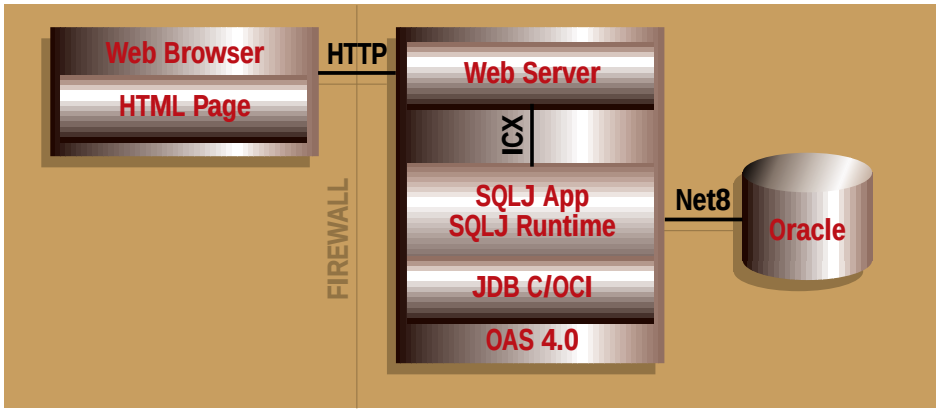


Figure 4: Using SQLJ in a three-tier architecture can be used to build "thin" browser-based applications with SQLJ deployed on the mid-tier.

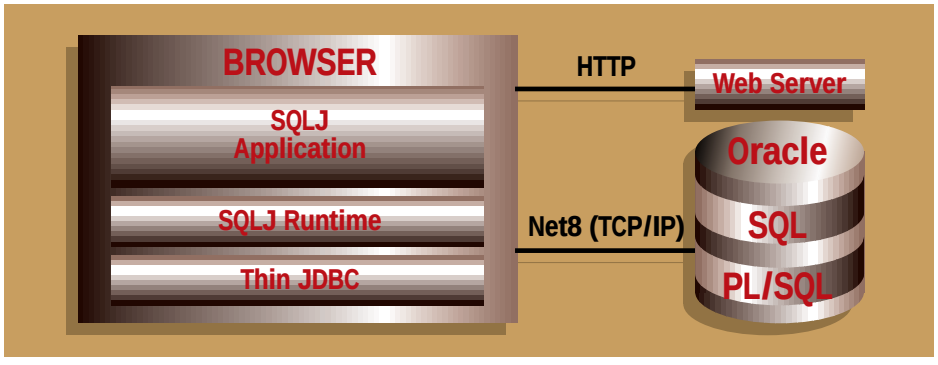


Figure 5: Using SQLJ with thin clients

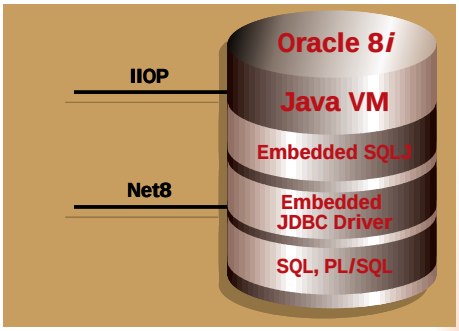


Figure 6: Using SQLJ for database stored procedures

SQLJ Features

The following is a look at SQLJ's most important features.

SQLJ Clauses

Static SQL statements appear in an SQLJ program text as SQLJ clauses. An SQLJ clause is introduced by the token "#sql", and contains an SQL statement inside curly braces. An executable SQLJ clause may appear wherever a Java statement may appear. Here is an example of an SQLJ clause that contains a SQL UPDATE statement:

```
#sql { UPDATE TAB SET COL1 = :x WHERE COL2
> :y AND COL3 < :z };
```

Host Variables

The inputs and outputs of SQL statements are passed through host variables. A host variable is a Java variable, parameter or field that's embedded in an SQL statement and prefixed by a color. The standard JDBC types, such as Boolean, byte, short, int, String, byte[], java.sql.Date, Integer, Double, etc., are valid host variable types in SQLJ. In addition, Oracle's SQLJ Translator supports Oracle7- and Oracle8*i*-specific types, such as ROWID, CLOB and BLOB, as well as Object and REF types.

Listing 1 consists of two SQL table definitions and a Java method containing SQLJ clauses that access those tables. It shows that SQLJ is quite similar to the ANSI/ISO-Embedded SQL and allows SQL statements to appear directly in program logic. At program development time, static analysis can detect errors in their SQL syntax, in their uses of tables and other schema definitions, and in their numbers and types of arguments and results.

Result Sets Returned by Queries

In an SQLJ program, a result set returned by a multirow query is manipulated by means of an iterator object, which iterates through the rows in the result set. An iterator is an object of an iterator class, which is a Java class defined by a declarative SQLJ clause that can appear wherever a class definition can appear. The clause defining a named iterator class lists the Java names and types for columns in a row of a result set. The following clause defines an iterator class called *AllStock*:

```
#sql iterator AllStock (String part, int
quantity);
```

This clause implicitly defines the Java class AllStock with methods named *part* and *quantity*, of types *String* and *int*, respectively. Those column-accessor methods return the values of columns from rows of a result set contained in an iterator of type AllStock. The following SQLJ program fragment defines a local variable of class AllStock, executes a query to populate that variable with an iterator object, calls the column-accessor methods of the iterator and prints the column values. The code for this can be found in Listing 2.

In addition, SQLJ provides support for defining positioned iterators that use traditional FETCH… INTO syntax to access query columns by position.

Database Connection Management

Listing 2 contains no explicit management of database connections. Its SQL statements execute on the default database connection, which is global to the program. SQLJ programs may also manipulate multiple database connections. Users may explicitly declare a connection-context class wherever a Java class declaration is permitted.

Combining Static and Dynamic SQL – SQLJ and JDBC

An SQLJ program may contain both SQLJ clauses and JDBC calls, for static and dynamic SQL, respectively. The two paradigms interoperate at the level of database connections and result sets/iterators. For example, an SQLJ connection context can be initialized with an existing JDBC connection:

```
java.sql.connection conn = …;
  PartsDb pdb = new PartsDB(conn);
```

It's also possible to extract a JDBC connection object from an SQLJ connection context. Similar conversions are possible between JDBC result sets and structured SQLJ iterators. For example:

```
AllStock iter;
  #sql iter = { SELECT … };
  java.sqlj.ResultSet rs = iter.getResult-
Set();
```

Thus the dynamic SQL API for SQLJ is JDBC.

SQLJ Code Versus JDBC Code

For SQL statements with input arguments, SQLJ clauses are often shorter than the equivalent dynamic SQL (JDBC) calls. This is because SQLJ uses host variables to pass arguments to SQL statements, while JDBC requires a separate statement to bind each argument and to retrieve each result. Contrast SQLJ and JDBC program fragments for the same single-row SELECT statement can be found in Listing 3.

Unlike dynamic SQL, SQLJ permits compile-time checking of the SQL syntax:

- Regarding the type compatibility of the host variables with the SQL statements in which they are used
- Pertaining to the correctness of the query itself, with respect to the definition of tables, views, stored procedures, etc., in the database schema.

Type- and schema-checking are also done where column data is fetched from an iterator object (by a FETCH statement or by column accessor methods). This is because the class of the iterator object defines the number and types of columns in rows contained by that iterator.

Using SQL Objects in SQLJ

Starting with Oracle8 (version 8.0), Oracle introduced support for abstract data types or objects. These objects are similar in nature to the traditional object-oriented classes or types. They have both attributes and methods associated with them. In Oracle8*i*, you can perform object manipulation through static SQL statements in SQLJ programs. Any kind of SQL data can be read and written by Java in a fully user-customizable fashion. Users can provide their own customized mapping from RAW columns in SQL to serialized Java objects. The same mechanism is then employed to create mappings from SQL object types to Java classes. This is done using the JPublisher tool, which assists in the generation of customized Java class definitions for these types.

SQLJ supports Oracle8 types through an Oracle-specific customization of the SQLJ runtime profile. This customization is performed automatically when you use the SQLJ translator provided with the Oracle8*i* database. In this process, runtime calls to standard JDBC entry points – such as getObject() and setObject() – are replaced with calls to Oracle's JDBC API.

We expect the SQLJ specification to evolve in the future and to encompass structured SQL3 types, such as those introduced in JDBC 2.0 and supported preliminarily in the Oracle8*i* JDBC drivers. Currently, support for Object Types can only be provided as a vendor-specific extension.

Oracle provides the JPublisher tool for automating much of the effort in creating the corresponding Java declarations for the Oracle8 object types and collections such as Object Types, REFs and Collection Types.

Java Stored Procedures

Java stored procedures are a part of the SQLJ standardization. Along with the embedded JDBC driver, the Oracle8*i* server also has an embedded SQLJ translator that allows application developers to write applications that access persistent data using SQLJ. Once you have written your Java program and tested it, you need to load it into Oracle8*i* – that is, onto the database's Java VM – and resolve all references. The database supports a variety of different forms in which Java programs can be loaded, including Java source text, standard Java .class files or Java archives (.jar). Java source loaded into the database is automatically compiled by the Java bytecode compiler hosted in the database. The Java programs loaded into the database are stored as "library units" in a database schema similar to how PL/SQL program units are stored in the database. Java library units needed to execute a Java program can be physically located in different database schemas.

Using SQLJ in Enterprise JavaBeans

The Oracle8*i* RDBMS is tightly integrated with the Oracle8*i* EJB server. This enables the Enterprise JavaBeans running in the database to use JDBC or SQLJ to access persistent data. Using Oracle8*i* session Beans, users can explicitly persist the state of their Bean and manage the Bean's persistent state via JDBC or SQLJ.

SQLJ in IDEs

SQLJ was designed so it could be embedded in Java IDEs. Oracle provides an SDK that enables tools and vendors to incorporate Oracle's SQLJ Translator into their Java tools. Oracle's premier Java Development tool, JDeveloper, was the first such tool to support SQLJ. JDeveloper has complete support for authoring and debugging SQLJ programs.

Summary of SQLJ's Benefits

SQLJ is a highly productive, open standard for embedded SQL in Java. It's supported by leading database vendors such as Oracle, IBM, Sybase and JavaSoft. SQLJ programs can be deployed in a number of different configurations, including two-tier client/server applications and three-tier intranet and extranet applications. They can write database stored procedures, triggers and methods with Oracle8*i*. Oracle's SQLJ Translator conforms to the SQLJ standard and provides support for a number of database features specific to Oracle. Oracle has achieved tight integration between SQLJ support in Oracle8*i* and JDeveloper for utmost speed and efficiency in generating bug-free code. The company has also given the user flexibility and choice from within the IDE to deploy the application in a variety of ways.

References

For further reading on the topics discussed above, please visit the Oracle Java home page at www.oracle.com/Java/

CODE LISTING

The complete code listing for this article can be located at **www.JavaDevelopersJournal.com**

About the Author

This first look at SQLJ was written by Oracle's SQLJ Development Team. For more information contact Moe Fardoost at MFARDOOS@us.oracle.com.

MFARDOOS@us.oracle.com

Java Technology for NFS:

Using an Internet File Access Protocol

by **Brent Callaghan**

Do your Java applets and servlets need to read and write files stored on a server elsewhere in the network? If so, you need NFS, a fast file-access protocol that is destined to become a standard for file access over local area networks (LANS) and the Internet.

Much of your data may already be stored on NFS servers, and now you can use NFS technology to read or write remote files directly from your Java program.

Naming Files on a Network and Getting to Files with NFS

Let's start at the beginning. The NFS protocol has been used for network file access since the advent of LANs in the early 1980s. Although the protocol is normally associated with Unix clients and servers, there are implementations available for almost every computing platform.

The NFS protocol is normally built into the operating system on either the client or server side. This leads to solid performance and enables programs to get to remote file systems as if they were local.

To access files on a network server, you need to be able to name the files using a network filename that will identify the file and its server from any computer in the network. We are already familiar with network names in the form of a URL. The URL Connection classes provide URL naming to identify Web pages.

You don't have to rewrite or recompile programs to use NFS, plus NFS files are commonly named through an automounter service that creates network names such as "/net/servername/dir/file." There is also an NFS URL, "nfs://servername/path."

Getting to NFS from Java

The Java JDK provides good support for Java as a network programming language. It's easy to connect to an HTTP server via the URL Connection to read Web pages or use the JDBC classes to access database servers. RMI provides a wonderful framework for network communication between Java objects.

However, the java.io classes provide no support for URL-style naming, for example, to read a file using a FileInputStream:

```
InputStream in = new
FileInputStream("D:\DATA\FILE");
```

The path name for the file must use the syntax for the underlying operating system. The OS may provide its own form of remote file access via CIFS or NFS protocols, but there is no consistent naming model that a user or programmer can rely upon. For example, the file "D:\DATA\FILE" may indeed be a remotefile since the "D" disk may be assigned to an NFS mount, but there is no reliable way to name files on other network servers as we can do with URLs and Web pages:

```
InputStream in = new
URLConnection("http://server/page").get-
InputStream();
```

Furthermore, the JDK provides no support for remote file access. When you write a Java program that reads or writes a file, you will use the classes in java.io. With these classes you can read and write files either sequentially or randomly. You can also list directories and create, delete and rename files. The java.io classes work well until you need access to files across the network – java.io provides no support for network file access.

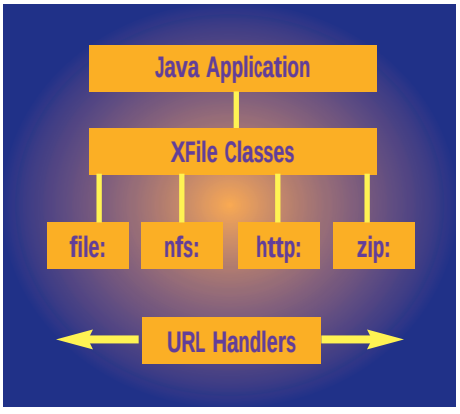


Table 1: The XFile classes provide a Java application with easy access to files through plug-in handlers identified by a URL scheme.

The NFS team at Sun Microsystems created a set of classes that enhance thej ava.io classes to allow URL naming. The XFile classes are almost identical to the java.io classes. They take the same arguments, return the same results and throw the same exceptions. Only the class names are different: an "X" is prepended to each class name. In addition to taking the same "native" file names asjava.io, the XFile classes will handle URL names. For instance, to test if a file exists:

```
XFile xf = new
XFile("nfs://server/a/b/c.txt");
if (xf.exists())
System.out.println("file exists");
```

Listing 1 demonstrates a simple program that uses the XFile classes to copy afile. It's identical to a java.io equivalent except for the "X" in front of the class name. The XFile classes provide this file copy program with a unique capability: rather than copy a file from one place to another on a local disk, the source and destination files can be named with URLs. For instance:

```
java xcopy  nfs://server/a/b/c   /tmp/x,

will copy a file from a remote NFS server
to local storage, and

java xcopy nfs://server1/a/b/c  nfs://serv-
er2/a/b/c
```

will copy a file from one NFS server to another. The XFile classes support an XFile Accessor interface that allows handlers to be written for any URL type. The NFS classes are just one type of handler for URLs that have the scheme name, "nfs:". Handlers can be written for other protocols, such as HTTP, FTP or CIFS. Handlers can also support other file system types. For instance, a "zip:" URL handler could be written that provides access to files within a zip archive.

Listing 2 is an example of an HTTP handler that implements the XFile Accessor

interface for XFile handlers. This handler provides access to Web pages as if they were files. Due to the limitations of the HTTP protocol, the handler cannot provide directory listings or random access to files. HTTP servers don't normally allow clients to create files or directories without the assistance of a customized CGI script.

Listing 3 is an enhanced copy program that will copy an entire tree of files from source to destination. Since the program needs to list directories, it can be used with local files and NFS, but not HTTP or FTP.

Getting Java NFS

You can download a zip file containing the XFile package and a handler for NFSURLs from www.sun.com/webnfs. The download contains documentation, javadocs for the classes and the classes themselves. The NFS handler needs only 70 KB of bytecode, yet it implements a capable NFS client. The classes achieve good read and write performance through the use of Java threads to implement read-ahead and write-behind techniques. In addition, the handler caches file attributes, directory listings and file data. This NFS client will interoperate with many different NFS server configurations: TCP or UDP connection, NFSversions 2.0 or 3.0 and the use of fast WebNFS connection or the MOUNTprotocol.

The XFile classes provide equivalent java.io access to a variety of file system types using URL naming. The bundled NFS handler provides convenient, run-anywhere access to files on your NFS servers, which are already widely deployed on TCP/IP intranets.

The NFS protocol is destined to become a standard for file access on the Internet. The WebNFS extensions to the NFS protocol have already made Internet NFSservers accessible from Web browsers and through corporate firewalls (see www.sun.com/-webnfs).

CODE LISTING

The complete code listing for this article can be located at **www.JavaDevelopersJournal.com**

About the Author

Brent has been with Sun for 12 years in the Solaris NetworkingTechnology Group working mostly on NFS. He participated in the development of the NFS version 3.0 protocol and the design and implementation of the Java NFS classes. He's currently co-chair of the NFS version 4.0 working group in the IETF. He can be reached at jeff.brainard@eng.sun.com.

jeff.brainard@eng.sun.com

JDJ FEEDBACK

JDJ's JBuilder Focus Issue *on a newsstand in* ***Milan, Italy****. Not all of the copies have CDs in them.*

JDJ Special CD in South Africa?

Dear ***JDJ***,

I am a software developer based in South Africa. I am keen on Java development and I purchased a copy of Inprise *JBuilder* in September last year. I was walking into the local news agent (CNA - Central News Agency) last week and I saw your special issue on *JBuilder*. It looked great and I took it off the rack. I was impressed by the *FREE* CD offer and went to customer information where CDs are normally collected, but was disappointed that no CD was available. I have since went to 3 branches of the CNA and at the last one spoke to the manager in charge of subscriptions. Apparently we (that is, South Africa) do not receive the CDs with all publications. Can you explain the policy of why the CD is not made available to South African readers?

Do I need to contact Inprise in this regard? If this is not the case how do I go about obtaining the CD?

I have made use of your offer to subscribe electronically. Thank you very much for what looks like a great publication.

Johan van Niekerk
jvniekerk@acenet.co.za

Dear Johan,

As with any issue of Java Developer's Journal, *when there's an announcement on the cover indicating that the particular issue is bundled with a CD, you are entitled to receive the magazine with the CD. The issue which you are referring to contained an Inprise* JBuilder *CD and was shipped to all newsstands with this CD included.*

There are two possible reasons why you could not find the CD in the magazine. They may have been ripped from the package and taken by a visitor to the newsstand or bookstore. That's a common scenario both here in the United States and on newsstands around the world. The JBuilder *CD was only available in that special issue of* JDJ *and made it especially sought after.*

Our spot checks of the newsstands proved that in most cases, this was the cause of why the CDs that were bundled with JDJ were missing. *(See above photo. This particular bookstore in Milan, Italy received 20 copies of our* JBuilder *special issue. When we spot-checked their racks in mid December, we found that 16 copies were sold, four copies were still available and three copies of the four were torn open and the CDs were gone.)*

Since you mentioned that none of the CNA stores had the CDs, this might be because of either the country's import regulations, or the CNA's policy of removing the CDs from the magazine and disbursing them via their cash register.

Either way, a new CD was mailed to you and you should have received it by the time you are reading this.

Mitchell Low
JDJ *Online Customer Service*

Stop Spamming Me!

Dear ***JDJ***,

With the start of the new year, I have been receiving e-mail notices from *JDJ*. I apologize for responding to a few of them without actually reading your entire e-mail message. I thought they were spam letters until I read the e-mail I received this morning, which delivered me a copy of *JDJ*'s February Digital Edition. Is it possible for you to send me the previous four e-mails I received and trashed without reading them? I don't want to miss one single issue and you guys are doing a real terrific job in providing the best Java news and information available out there.

John Hamilton
Schaumburg, IL
john.hamilton@iname.com

Dear John,

We redirected the e-mail that you missed in January back to you. Thank you for your support and kind words.

-The Editors

eb Site

persJournal.com

SYS-CON RADIO INTERVIEW

Broadcast live at the Java Business Expo in the Jacob Javits Center in New York City, SYS-CON Radio's Chad Sitler spoke with Michael Gardner and Joe Nicholson of Inprise, and Michael Smith of BEA WebXpress

michael gardner
director of development for Java products
joe nicholson
director of marketing enterprise tools
Inprise Corporation

JDJ: *We know that Inprise has focused largely on Java with JBuilder. What else are you focusing on?*
Nicholson: We're very focused on the business aspects of what companies are doing, and IT has really become a focus for how companies could be more competitive. You know, Java has a way of fundamentally changing the way people do IT, and so we're really looking at Java as the key component of the Inprise strategy, not only in the JBuilder IDE tool but also across all our products. Java is key to Inprise's long-term strategy because we believe in the fundamental business aspects and advantages of Java. We know that the ability to take Java code and put it on any platform throughout the enterprise is very important to companies.

JDJ: *Could you tell us a little about the overall enterprise Java strategy at Inprise?*
Nicholson: Sure. We're using Java as a fundamental development environment for an entire suite of tools. You know, part of what Inprise is doing is addressing the entire workflow process in applications, from development to deployment to integration and management, and we're using Java as a fundamental language and platform to develop tools across that entire spectrum.

JDJ: *We've got a lot of vendors here at the Expo. What sets Inprise apart?*
Nicholson: That's a good question. We take a fundamentally different approach. Our focus is on the entire development spectrum and being able to address what the customer needs, from development through deployment, through the integration with Legacy applications, on through to the actual live management of the application. And we're doing that in ways that are very important to customers. For one, we're completely standard-neutral, that is, we support every major standard in the marketplace. We're also platform-neutral in that we have Windows environments supporting common CORBA. We have Java environments supporting CORBA and we also run on a variety of UNIX machines. What we're about is what the customer needs, not necessarily what our particular IT agenda happens to be.
Gardner: We have a deep, deep, deep understanding, and have for years, of up-storing the computing of languages and [with the acquisition of Visigenic that was completed in early 1998] with distributed object computing. And it's the depth of experience and understanding of Java and the role Java can play in object-oriented computing that our personnel has that I think is very unique.
Nicholson: We're approaching things more from a business perspective, rather than simply from an IT perspective. We have a rich history in simplifying difficult development tasks, whether it was client/server development with Windows or Java development with JBuilder. We're going to do the same thing with the Inprise Application Server. This application server environment radically simplifies the often difficult task of writing three-tier applications. When you hear from customers, what they want to know is, "How can I get my application to market quicker?" "How can I make sure it's robust?" and "How can I make sure it's scalable?" and we're giving them the tools to do that.

JDJ: *What do you have in the works? What's in the future for Inprise?*
Nicholson: Well, we have lots of things in the works, such as continuing to enhance the integration of our IDEs with the application server we announced [in early December 1998]. In addition to having the Java platform supporting it, we'll also be supporting it on both our C++ and our Delphi tools. We have a variety of initiatives going on in Java for future releases of JBuilder. And again, if you look at what we are from a company perspective and a company strategy perspective, as application development continues to be more and more complicated, we're trying to make it all simpler so companies can extract the business benefits out of both Java and IT.

JDJ: *What do you suggest for those companies as far as going about finding these solutions?*
Nicholson: Well, I think probably the first thing is to get started today. In the case of our tools, you can begin working on JBuilder today and be compatible with the code that JBuilder ultimately produces because it's 100% Pure Java. So regardless of where companies want to go in the future, you know that the code you're producing in JBuilder today is 100% Pure Java and you can get all the benefits of being able to move that code throughout your organization, both on the client and on the server.
Gardner: But I also think that people have been in evaluation mode about Java. We've talked to customers who bought JBuilder or another proven tool and who have written some experimental applications but they haven't really moved – certainly some customers have, but in general I'd say that people have been learning about it but they haven't really committed to actually using it yet. I think with the release of Java 2 it's now ready, and so part of getting started now, I think, is actually realizing that Java is here and it's real and there's no reason you can't bet on it. And I think that's an aspect of what we're going to start seeing now with the release of Java 2.
Nicholson: The other aspect is that we're also now seeing the early adopters of Java technology really starting to reap the benefits of what Java can do both cross-platform and around their organizations. And I think over the next few months as the Java 2 platform really starts to be implemented, we're going to see some real success stories in terms of quantifiable benefits. We're basically committed to Java across our entire company because we see the business benefits and the return on investment that we just haven't seen with other operating and platform systems in the past. So I think it's a pretty exciting time to be in the marketplace.

michael smith
senior systems engineer
BEA WebXpress Division

JDJ: *Of the products and services BEA offers, above and beyond those of your competitors, what would be the most appealing to a consumer?*
Smith: We talk a lot about that, and I think what we have at BEA is a philosophy: when you buy software, a tool or especially middleware – which is what BEA focuses on – you're really buying a relationship. We feel we deliver what we promise, support what we promise and carry you all the way through. So in that regard, we think we offer people a very comprehensive set of middleware platforms that's unrivaled in the industry. Couple that with our focus on the Java technologies, as well as with WebLogic being the first server with an EJB deployment, and we feel end to end that we operate a relationship. And we feel that it's those relationships, every bit as much as the technology, that make us an important company.

JDJ: *What do you see for your company and your products in the immediate future?*
Smith: I think BEA is focusing on delivering the message for what their solution offers. I think we're working on a lot of neat things that we really haven't evangelized enough in the marketplace. So I really think we're focusing on telling our story, telling our relationships. And the big thing right now, obviously, is that the WebXpress division has to get the WebLogic 4.0 product out the door. The second phase of that is the implementation of a couple of the new Java enterprise APIs. JMS, the Java Messaging System, comes to mind. The other thing that's really important to the BEA product line is the integration of all our products. We want to make sure that our whole product line speaks together very well, and that developers have tools that work with all these things. We're working really close with Symantec right now and we're getting an integrated version of Café that allows developers to not only build the GUI but actually to build the client-side code as well, and deploy it all within the same developer infrastructure.

JDJ: *What kind of audience are you targeting for these products?*
Smith: I think we're definitely targeting the enterprise – people wanting to do large-scale deployments, and technologies using the Internet, Java and middleware platforms.

About the Author
Chad Sitler, host of SYS-CON Radio, is SYS-CON Interactive's senior Web designer. Chad can be reached at chad@sys-con.com.

Enterprise Solutions Conference

www.jumpstart99.com

Java NT Services

Migrating your Java server from UNIX to an NT boot-time environment

by **Jim Barnebee**

As the popularity of Java increases, many information services departments are embracing Java as the solution to their cross-platform challenges. As this trend progresses, many developers will be faced with the challenge of migrating their Java servers from UNIX to an NT boot-time environment. Administrators tend to take the attitude that since Java is a cross-platform language, this migration should be effortless. Unfortunately, this isn't the case. The thread management of child threads in the current Java Virtual Machine from Microsoft can cause problems for those developers migrating from a UNIX environment. This article explores one workaround for the thread management and garbage collection differences between the Sun and Microsoft JVMs. Microsoft recently agreed to implement a court decision to make their JVM ISO-compliant. It remains to be seen whether this will affect the behavior of Microsoft's proprietary Service class.

Considerable trial and error is associated with the migration of Java server classes to the NT environment. In the UNIX operating system it's possible simply to fork off a Java process in a script that's run at boot time. (The particular script, and the associated operating system messages that need to be caught, depends on the flavor of UNIX involved.) Under the NT operating system no fork is available, so to start a process at boot time, an NT service must be created and registered with the service's control panel. The problem inherent in this approach is that the NT OS doesn't allow fine control over the child process once it's been started. If a Java class utilizes threads, you have to override the stop method of the extended Service class to isolate and stop your threads manually. This task is made more complex because there's no definitive way to isolate your threads from the operating system threads other than their placement on the stack. Consequently, other processes started after your server may interfere with the stopping of your service. Recent versions of the Microsoft JVM have addressed this problem, but to ensure proper garbage collection, manual thread management within services is still necessary.

To create an NT service, you have to create a Java class file that extends the coms.ms.service.Service class provided by Microsoft in its Software Developers Kit for Java. This class is located in C:\SDK\NTService\service.zip in version 1.02 and C:\SDK-Java.202\Bin\jntsvc\service.zip in version 2.02 of the SDK files installed by the Microsoft SDK-Java installer. *Do not attempt to unzip this file and examine the Java files.* All the documentation you should need to extend the Service class is contained in the SDK-Java documents and in this article. The SDK-Java is available on Microsoft's Web site at www.microsoft.com\java\. This executable should be downloaded and installed on the system that will run the Java service. Be sure to install the Internet Explorer Java support as part of the SDK installation because the support actually installs the Java Virtual Machine. The JVM is necessary because the NT operating system won't try to access the normal Java executable located on the path; instead, it will invoke the jview executable resident within the Microsoft SDK. Microsoft's proprietary compiler and JVM contain custom hooks into the OS that are not part of the ISO standard. It remains to be seen whether these hooks will be deprecated as part of the move toward ISO compliance by Microsoft. The services control panel now defaults to the Microsoft proprietary JVM, called jview.

> ***"Microsoft recently agreed to implement a court decision to make their JVM ISO-compliant."***

The Service class of the coms.ms.service package provides methods to allow the NT OS to start, stop and pause a service. However, the JVM doesn't allow the NT services driver to access child threads of the parent class automatically. The ramifications of this are that if you create a wrapper class that extends service, you have to manually instantiate, start, pause and stop any child threads from within the wrapper class itself. Any child threads instantiated or started from within the Java class being wrappered won't be accessible from within the NT services framework. Once a child thread has been instantiated and started, the NT OS no longer knows that the child thread is grouped with its parent. Any attempts to stop or pause a service that contains child threads without taking this into consideration will result in the OS's returning an error stating that it is unable to stop the service. To avoid this, the wrapper class in its stop method has to isolate and stop any child threads. Note that this is true for wrapper classes, while classes that directly extend the coms.ms.service.Service class will stop, usually without error, in version 2.02 and above. For previous versions, and to ensure proper functionality if wrappering classes are involved, manual thread grouping should still be utilized.

Listing 1 is an example of how you might extend the coms.ms.service.Service class to create a wrapper class that can be registered as an NT service, and will call the public static void main(String[] args) method of your Java class. To use this example code, replace YOURSERVICE with the class name of the application you wish to wrapper.

Once your Java file has been created, it needs to be compiled to bytecode. Once you have a Linking.class, move the Linking.class file to the C:\SDK-Java\NTService\ or C:\SDK-Java.202\Bin\jntsvc\ directory that was created when you ran the Microsoft SDK-Java executable. You now have a wrapper class that can be registered as an NT service.

To ensure that your target system is configured properly so you can register and use the Java service class you've created, follow the steps below.

1. Make sure there is a local (on hard drive) copy of any classes used or imported by the YOURSERVICE class; NT services aren't started in any particular order, and the networking connections may or may not be functional when the JVM attempts to start the Linking.class.
2. Place the service.zip file and the path to the tree of classes imported by YOURSELF in the CLASSPATH environmental variable in the system environment settings. (This file is located in C:\SDK\NTService\service.zip in version 1.02 and in C:\SDK-Java. 202\Bin\jntsvc\service.zip in version 2.02 of the SDK.) These settings can be assessed by opening the Control Panels Dialog from the Start menu and double-clicking on the System icon, which will bring up the system parameters. Clicking on the Environment Tab will access both the systemwide and the individual user parameters. You'll need to modify the system CLASSPATH parameter.
3. (If you're using version 2.02 or higher of the SDK, you can skip this step as the jntsvc registration program will modify the registry for your service.) The last step is to modify the JVM's CLASSPATH variable, because the standard Java CLASSPATH environmental variable isn't recognized by the JVM. The JVM CLASSPATH variable is explicitly defined within the JVM's set of environmental variables. Therefore it's necessary to directly modify the CLASSPATH variable that the JVM places in the system registry as follows:
   - Access the MS-DOS Command Prompt from the Start menu. In the Command Prompt window type regedit <return> – this brings up the system registry.
   - Open the folder HKey_Local_Machine by clicking on the plus (+) sign next to the folder. Open the subfolder software and the JVM folder in the same manner. The path in the bottom left-hand corner of the System Registry Editor window should now read:

```
My Computer\HKey_Local_Machine\Software
\Microsoft\JavaVM
```

Now open the CLASSPATH environmental variable for editing by double-clicking on the small book icon directly left of the word CLASSPATH.

- Then remove the period at the end of the current entry and add to the current path the path to the class tree that YOURSERVICE imports. Make sure that C:\SDK-Java\NTService\service.zip and C:\SDK-Java\NTService\ are included in the CLASSPATH. Path additions in the NT OS are separated by a ;. Remove the period at the end of the CLASSPATH before making additions, and don't add any unnecessary punctuation marks!
- Close the System Registry Editor.

The final step in the migration process is to register the service class with the NT Services Control Panel.

Registering the Service

For version 1.02 (this is also explained in the SDK-Java documentation):

- In the C:\SDK-Java\NTService\i386\ directory run the svcsetup program with the command line svcsetup ServiceName Linking -classpath C:\SDK- Java\NTService\.

> *"Many companies are pursuing Java in an attempt to reduce their operating budgets and increase their efficiency."*

- From the Start menu open the control panels, then the Services Control Panel.
- Select the ServiceName Service and click on the STARTUP button; select automatic, and allow interaction with the desktop.
- Select HWprofiles, and select enable.
- Reboot your computer.

For version 2.02:

- In the C:\SDK-2.02\Bin\jntsvc\ directory run the jntsvc program with the command line jntsvc /out:ServiceName.exe/main:ServiceName ServiceName.class.
- Run ServiceName.exe\install to register the Service.
- Further documentation on use of the jntsvc should be available at C:\SDK-Java.202\Docs\SDKJDoc\def_cerv.htm provided that you installed the SDK to the default root.
- From the Start menu open the control panels, then the Services Control Panel.
- Select the ServiceName Service and click on the STARTUP button; select automatic, and allow interaction with the desktop.
- Select HWprofiles, and select enable.
- Reboot your computer.

Testing and Evaluation

There are two ways to see whether these procedures have been successful. The first and easiest is to open the Control Panel again and see if the service has started. If so, you've been successful. If not, an error has occurred. You can bring up the Event Viewer for the NT OS by looking under the Start menu's Administrative Tools (common) area and selecting Event Viewer. Set the Events to the Application log, and refresh the view. This allows you to examine the debug lines from the linking class so you can debug your classes or linking structures. *Note:* If the Event Viewer Application log says it can't find your class, make sure both CLASSPATH variables are set to include the paths to your linking class and all classes you wish to import.

Many companies are pursuing Java in an attempt to reduce their operating budgets and increase their efficiency. Java is promising cross-platform compatibility. Most of the problems associated with the migration of Java can be attributed to its youth as a language. While developers feel joy about a "write once, run anywhere" ability, it's short-lived when the virtual machines don't conform to the same specifications. While Sun has made – and continues to pursue – true cross-platform compatibility among all JVMs, not all have achieved a workable level of compliance. As a result of litigation, Microsoft is migrating its JVM into compliance with the ISO standard. It remains to be seen, however, whether this migration will affect the necessarily proprietary NT Service installation classes. These differences require developers and IT staff who wish to import their code to Windows to be aware that the "write once, run anywhere" promise of Java is not always applicable in a Windows environment. Until all Java Virtual Machines behave identically, there will continue to be issues of migration for developers and IS departments to face.

CODE LISTING

The complete code listing for this article can be located at **www.JavaDevelopersJournal.com**

About the Author

Jim Barnebee, a registered Sun developer and a Microsoft developer, has been an alpha and beta tester for many Java development environments. He has a BS in information systems science, and has been developing software professionally since 1986. Jim can be reached at philosopher@stones.com.

philosopher@stones.com

The JConfigure Widget

This flexible, time-saving widget can be a blessing to users and programmers alike

by **Claude Duguay**

One of the most common development requirements is the ability to store and modify configuration parameters. Java provides a Properties class that's useful for storing application settings. To make these properties easy for users to modify, programmers typically build a dialog interface that exposes each of the various fields and values individually. But this can be complicated, and often leads to a system that requires time-consuming code changes when variables are added or removed.

This month, we'll try to alleviate some of these problems with the JConfigure widget. This component automates most of the work you'd normally have to do and uses an open architecture to make it as flexible as possible. Our widget presents users with an explorer-style interface, allowing them to edit property fields based on a hiearchy of information. Figure 1 shows JConfigure in action.

By default, the fields are presented using JTextField editors, but a template mechanism allows you to define custom field editors by implementing a simple PropertyField interface. JConfigure is entirely data-driven and adapts to handle new fields or property files without typically requiring any code changes. The only code you should need to write will be for custom edit fields.

Application settings come in many forms. Under Windows, for example, they are commonly held in .ini files or in the Windows registry. Under UNIX you'll often find them stored as environment variables or .conf files. With Java programs, the standard choice is a .properties file. Java Properties use a dot-delimited path convention that supports a hierarchical name space which we can automatically parse into a tree structure.

User interface designs have started moving toward an explorer-style view for property settings. The older – and still perfectly valid – style was to use tabbed panels. This is still one of the best ways to present settings to the user, but it may become cluttered if you have too many categories. Explorer-style tree navigation keeps the interface well organized, supporting easy viewing and editing of almost any number of categories and settings.

Tree Modeling

Each of the entries in a properties file can represent a path with each path element delimited by a dot (period) character. We consider the last element to be the field name, so the path we're interested in contains all but that last element. We use the JTree control to display the resulting hierarchy. Listing 1 shows ConfigureTree, which merely subclasses JTree, making it easier to set the default size and model.

The ConfigureTreeModel class in Listing 2 shows how we extend the JFC DefaultTreeModel and add the ability to set and retrieve entire paths. This makes it easier to manage dot-delimited paths rather than the individual elements. The constructor sets "properties" as the root node. The addPath method adds child nodes by traversing existing nodes and appending new ones to the child list when appropriate. We use the StringTokenizer class to split the path into pieces and do each lookup, adding any necessary tree nodes as we traverse the path.

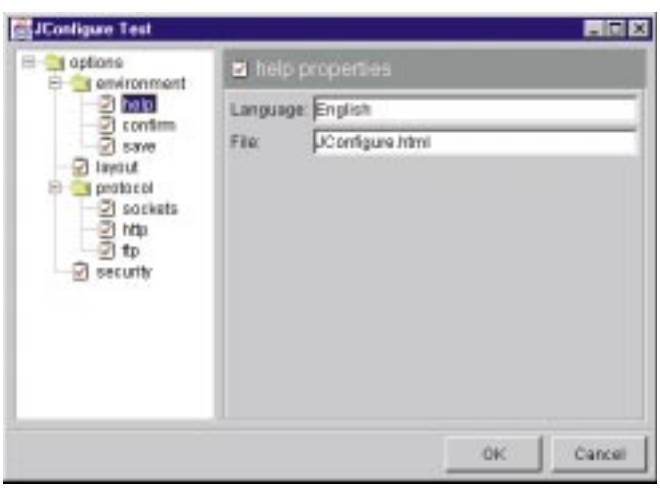


Figure 1: JConfigure at work

The ConfigureTreeModel also implements the getPath method, which returns a linear list of paths in the same order they're stored in the tree model. We do this by traversing the nodes, accumulating path segments along the way. The paths are returned in the form of a vector. We always maintain order when loading and saving property files so the presentation can be controlled more effectively.

Listing 3 shows the ConfigureTreeNode class. These are the nodes we use in the ConfigureModel to store path elements. We extend the DefaultMutableTreeNode class and add the ability to get the node name and return a child node by name. The getName method casts the user object back into a string to save time when we need to access it. The getChild method returns a ConfigureTreeNode by taking the name argument and looking at each child node until a match is found.

View Layout

Figure 2 shows the way panels and layout managers are brought together to work our magic. We've already talked about the tree, model and node classes. Listing 4 shows the TitleBar class, which is nothing more than an extended JLabel class with a few presets to control the font, spacing and colors. We use this class to display the current panel name, making the output context easier to understand.

The DeckLayout manager was covered in an earlier article, so I'll keep this brief. The class essentially replicates the code from the Java CardLayout manager, updating deprecated calls to their modern equivalents and disabling or enabling child components as required. This fixes a few shortcomings in the CardLayout manager which inappropriately allows focus traversal through invisible components. The net effect is much cleaner and requires no effort on the programmer's part in order to manage proper focus traversal.

We develop a new layout manager named FieldLayout to handle label/field pairing. You may find this layout manager useful in many of your own applications, since numer-

SIGS Conferences

www.sigs.com

ous programs have to deal with the same issues. FieldLayout expects prompt and editor component pairs, though you can use it with any valid component. We extend the AbstractLayout class (which was presented in earlier articles) cutting down on the amount of coding we need to do.

The FieldLayout code, presented in Listing 5, implements the minimumLayoutSize, preferredLayoutSize and layoutComponent methods. The first two merely calculate the required size, while layoutComponent does the actual layout work. In each case we calculate the maximum label width and consider the two-column format of our display to be directed by that value.

Accounting for the margin and the horizontal gap, we resize each component to the maximum label width in the left column and the remaining available width in the right column. The vertical size, accounting for the margin and vertical gap, is the maximum height of the two components on a given line. We also account for situations where an odd number of components are entered and no right component exists for a given row.

While the FieldLayout manager is completely generic (making no assumptions about the components displayed), the FieldPanel provides specific methods for presenting field information. Listing 6 shows the FieldPanel code. We set the layout manager to FieldLayout with vertical and horizontal gaps, and add an EmptyBorder to help with spacing.

The FieldPanel implements two addField methods. Each one creates a JLabel control with the field name provided in the first argument. The second argument is either a string (in which case we create a JTextField element), or a Component argument, which permits additional flexibility. We use this variation to insert custom editors.

FieldPanel implements two additional methods to retrieve information when we need to save the data. The getFieldNames method returns a string array with each of the field names. The getFieldValues returns a string array with the field values. The two arrays are guaranteed to be returned in the same order they were created so we can reconstruct the properties file when it's time to save it.

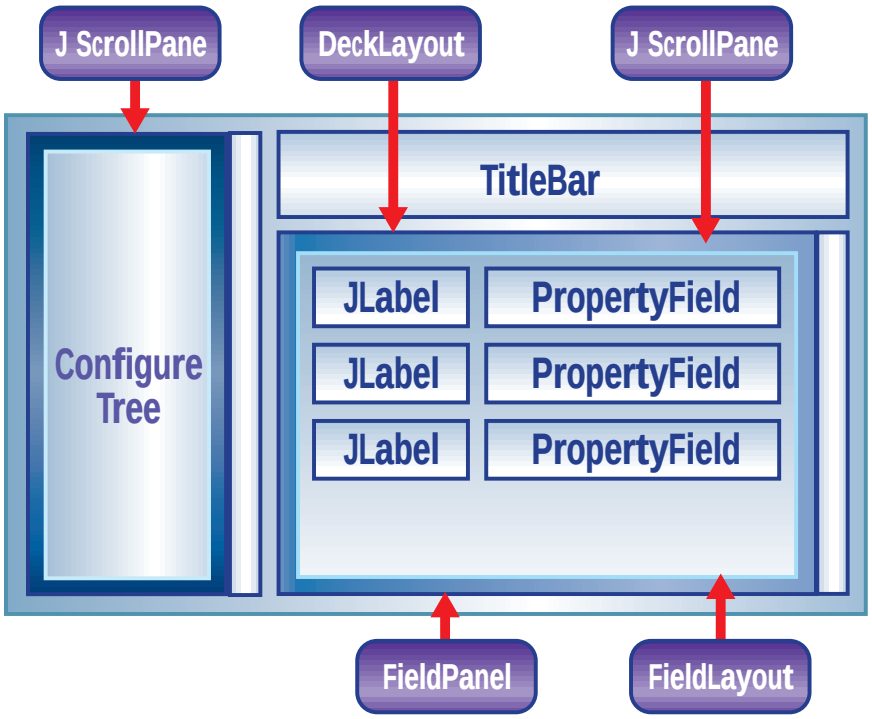


Figure 2: The JConfigure layout

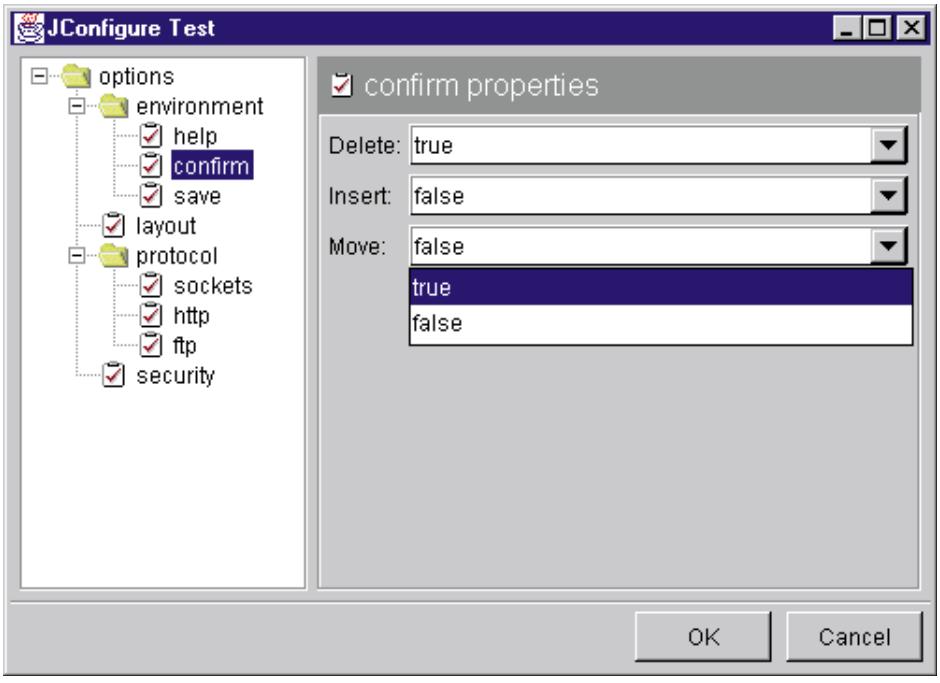


Figure 3: Custom Boolean fields

Custom Editors

To maximize the flexibility of our design, we support customizable edit fields. Listing 7 shows the PropertyField interface each must implement. The LayoutPanel is smart enough to recognize PropetyField components and uses their getValue method to retrieve values. The JConfigure class handles instantiation. The PropertyField interface must be implemented by a Component class that exposes the setValue and getValue methods, along with a zero-argument constructor.

We implement the obvious default PropertyField class in Listing 8. The PropertyTextField class extends JTextField and simply adds the setValue and getValue methods by calling the setText and getText methods in JTextField. Listing 9 shows the PropertyBooleanField class, which extends JComboBox and sets the values to true or false. The setValue and getValue methods use the JComboBox setSelectedItem and getSelectedItem to expose the required interface. Figure 3 shows what these editors look like.

The two previous classes show how easy it is to implement PropertyField classes. Listing 10 shows another – PropertyRectangleField – to demonstrate how you can create complex field editors with relative ease. The PropertyRectangleField class uses four JLabel and four JTextField components arranged in a 4x2 GridLayout on a JPanel. The setValue and getValue methods, implementing the PropertyField interface, set the

JTextField values based on the parsed comma-delimited property string and return an appropriate string based on the values in those text fields. Figure 4 shows a trio of rectangle fields in JConfigure.

The JConfigure Class

The JConfigure class ties all the previously mentioned classes together. The code is presented in Listing 11. The constructor creates the components we'll need to display our tree navigator, title bar and field panels. The properties file name is the only argument you need to provide. JConfigure will automatically load and parse the properties and display the interface for you. If a file with the same prefix and a .template extension is found, it'll also be read automatically.

The constructor code sets the leaf node icon for the tree and creates a BevelBorder to place around the title bar. The colors are explicitly set to keep the border thin, setting the inner part of the two-pixel border to the same color as the background. We set the panel border to an EmptyBorder with five pixels all the way around. We add the ConfigureTree to the WEST and a JPanel in the CENTER part of the panel's BorderLayout. The CENTER panel also adds a TitleBar to the NORTH and a new JPanel with the DeckLayout in its own center. Figure 2 shows how the internal components are organized.

We call on a pair of methods to read the template and property files. The readTemplate method merely opens a stream and uses the Properties object load method to read the content into a Hashtable before closing the stream. The readProperties method is more complicated because we want to preserve the order of elements in our properties file. The readProperties method expects a reference to the ConfigureTree components and the file name to be read.

Like readTemplate, readProperties delegates much of its work to another method. We open a BufferedReader stream and read each line independently before closing the file. Each line is parsed by the processLine method and the data is inserted in appropriate locations. We use the ConfigureTree addPath method to keep the tree model up to date and use an addField method to deal with each path tail and associated value.

Because we have multiple cards in the DeckLayout panel, we keep track of the path each one is associated with by using a Hashtable, assigned to the deck member variable. If a panel already exists for a given path, we simply add a field – otherwise we add another FieldPanel to the deck. As each field is processed, we check to see if there's a template entry associated with it. After creating a field prompt, we set the PropertyField. This is based on either the template entry we found or the default PropertyTextField. This late binding is accomplished at runtime, using the Class.forName(name).newInstance() combination.

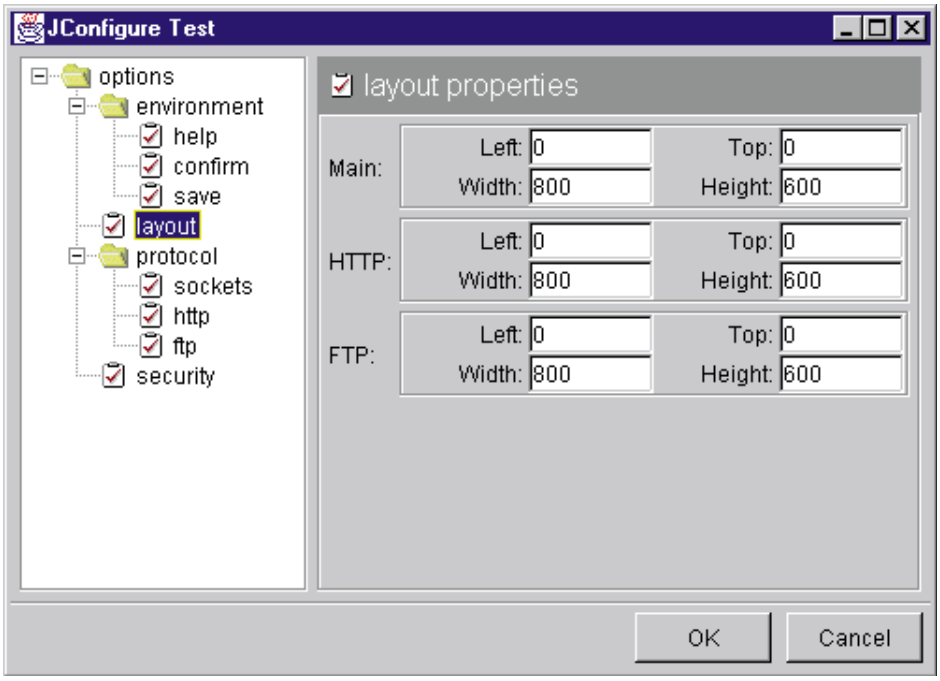


Figure 4: Custom rectangle fields

The JConfigure class exposes a pair of save methods. The first assumes you want to save the file to the same file name used at construction time. For integrity reasons, we first rename the existing properties file, changing the extension to .previous. This gives us a rollback position if something goes wrong. The alternative save method lets you specify the output file name and doesn't provide this extra safety feature (assuming the original is still available for recovery purposes).

The output file content is labeled with a JConfigure statement that includes the current date and time, formatted by the DateFormat class. We then walk the tree model paths after asking for them with the ConfigureTree getPath method. This is guaranteed to be in the right order but may include subpaths that have no associated FieldPanel, so we check each one and then walk the elements in each FieldPanel using the getFieldNames and getFieldValues methods. In each case, we rebuild the full path and write it to the BufferedWriter until we run out of properties.

To switch between views, the JConfigure panel is registered as a TreeSelectionListener. To support this, the valueChanged method responds to user selections by calling getSelectionPath and activating any existing FieldPanel that's stored in the deck Hashtable under that path. If no path exists, we leave the existing FieldPanel where it is. We use a final method called treePath to rebuild the path from the TreePath structure provided by the JFC.

Listing 12 shows the JConfigureTest class, which wraps a JFrame around the JConfigure component and adds an OK and a Cancel button. If the user clicks the OK button, we call the JConfigure save method. If the user clicks Cancel, we ignore the changes and save nothing. This is representative of the way you'd typically use JConfigure. Since JConfigure extends the JPanel class, it can easily be used in any window or dialog box.

Summary

The JConfigure widget provides considerable flexibility when it comes to manipulating property files. These files often represent the persistent states and customized settings implemented by your software. Having the ability to present an adaptive generic interface to the user is attractive. The user can automatically navigate and edit properties without a big investment on your part. Furthermore, the model is extensible. You can provide custom editors for any field and support constraints or field validation through a very simple interface. Enjoy.

CODE LISTING

The complete code listing for this article can be located at **www.JavaDevelopersJournal.com**

About the Author

Claude Duguay has been programming since 1980. In 1988 he founded LogiCraft Corporation, and he currently leads the development team at Atrieva Corp. You can contact him with questions and comments at claude@atrieva.com.

claude@atrieva.com

Raima Releases Velocis Database Server 2.1

(Seattle, WA) – Raima Corporation has released Velocis Database Server 2.1, which includes new interfaces for multiple development environments, including Rogue Wave's DBTools.h++, Perl, Java and Delphi. Other significant improvements include an integrated repair tool called "dbrepair" and two customized functions: support for scrollable cursors; and an extension to ANSI SQL called "customized comparison functions," which supports a development of sophisticated multilingual or "Soundex" features.

If it's fast - it's RAIMA High Performance Database Technology

A free trial version of Velocis 2.1 is now available for download from Raima's Web site at www.raima.com.

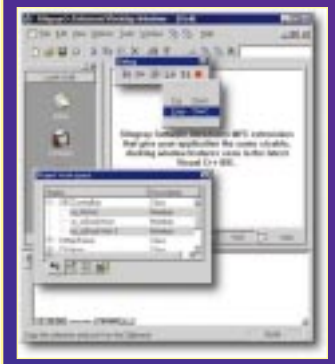
Gefion Software Releases InstantOnline Basic 2.0

(Hermosa Beach, CA) – Gefion Software has released InstantOnline Basic 2.0, its standards-based, cross-platform component suite for the development of dynamic, interactive Web sites.

InstantOnline Basic is a set of Java Servlet components for the development of Web applications. In addition to the database access and HTML presentation components found in 1.0, InstantOnline Basic 2.0 adds components for file uploading and server-side file manipulation, dynamic Web forms, sending e-mail, input validation and user session data handling. Other improvements are extended variable handling, debug features and enhanced error handling.

Gefion software

For more information contact Hans Bergsten at hans-@gefionsoftware.com, or visit www.gefionsoftware.com.



Rogue Wave Software Announces Objective Toolkit for WFC 2.0

(Research Triangle Park, NC) – The Stingray division of Rogue Wave Software, Inc., has announced Objective Toolkit for WFC 2.0, which includes robust support for advanced docking windows and toolbars, enabling developers to add enhanced views and controls to their applications.

Objective Toolkit's docking windows support creates a powerful framework that makes any component dockable and any area a docking target. The docking toolbar support builds on the docking window support, allowing toolbars to be docked along any side of the application windows or even float as a separate window. Objective Toolkit for WFC 2.0 also boasts several additional control enhancements to an already powerful set of complex GUI components and advanced frameworks, all tightly integrated with Microsoft's Visual J++ development environment.

For more information call Rogue Wave Software at 800 487-3217, e-mail sales@roguewave.com or visit www.roguewave.com.

Petronio Technology Group Offers Java Programming Courses

(Boston, MA) – The Petronio Technology Group has announced two new public Java programming courses:

- Boston, MA – March 1–5, 1999 – Ultimate Java Workshop
- Boston, MA – April 5–9, 1999 – Advanced Java Workshop

These have been added to the list of other on-site workshops offered by Petronio:

PETRONIO Technology Group

- Moving from Java 1.1 to Java 1.2
- JavaBeans Programming Workshop
- Ultimate Java Programming (for non-C, C or C++ developers)
- Advanced Java Workshop (includes CORBA programming)
- Object-Oriented Analysis and Design
- Object-Oriented Design Patterns (in C++ or Java)

To register or for more information about these workshops and the prerequisites for participating, please visit www.petronio.com/pubtrain.shtml, or call 781 778-2000.

Inprise Announces Support for JINI Technology in Future Versions of JBuilder

(San Francisco, CA) -- Inprise Corporation announced that it will provide support for JINI, Sun's connected computing technology, in future versions of its award-winning JBuilder family of Java development tools. This support will allow Java programmers to create JINI technology-based applications.

JBuilder is Inprise Corporation's award-winning family of comprehensive visual development tools for creating pure Java business and database applications for the enterprise. The JBuilder product family features fast and easy JavaBean component creation, a scalable database architecture, robust visual "Two-Way" development tools and the ability to produce "100% Pure Java" platform-independent applications, applets, servlets and JavaBeans. The product's open environment supports the Java 2 platform, JDK 1.1.6, JFC/Swing components, JavaBeans, Enterprise JavaBeans, CORBA, RMI, JDBC and all major corporate database servers.

INPRISE

For more information contact Inprise at 800 233-2444 or visit the JBuilder Web site at www.inprise.com/jbuilder/.

PageMart & ObjectSpace Profit from Strategic Partnership

(Dallas, TX) – ObjectSpace recently announced the results of its ongoing partnership with PageMart Wireless, Inc. The partnership resulted in the development of an advanced technology platform to support PageMart's narrowband PCS network for the advanced messaging industry. This network provides PageMart's customers with coast-to-coast connectivity on the world's largest, single-company-owned terrestrial two-way wireless data network.

OBJECTSPACE

The enhanced technology platform is based on a scalable, distributed architecture developed by ObjectSpace's Solutions Consulting team specifically for PageMart. The object-based system integrates a range of advanced technologies, uses the Smalltalk, C++ and Java programming languages, and was codeveloped using an interactive/incremental development process. This approach reduced risk and allowed a quick response to changing project needs and requirements.

PageMart Wireless

For more information, visit www.objectspace.com/news/press/company/01-19-99.html.

SIGS Conference

www.sigs.com

Java – Into Its 4th Year

Part 2 – A review of the distributed computing environment for Java

THE GRIND

by **Java George**

"In 1997, if you weren't engaged in IDL, RMI, CORBA, IIOP and Java, you might as well have been living in a cave."

Networked Applications Bring Distributed Computing to Life

Never has distributed computing been as commonplace as it is in these days of Web applications or Networked Applications. The requirements on such applications for distributed processing across multiple servers has brought distributed objects into the mainstream.

In the beginning of Java there was the home-grown approach. Certainly, there are plenty who have ventured down the road of establishing a TCP-IP socket from a Java client in order to remote control a related Java process on a server to get remote processing. We were all relieved at how simple the socket management was with this language and how networking was an obvious part of Java's lingua franca.

In the second year of Java we saw a surge on two fronts: Remote Method Invocation (RMI) and Java for CORBA (IONA's OrbixWeb and the Visigenic ORB). Early adopters moved quickly to utilize these infrastructures for *n*-tier computing and distributed objects, while authors such as Bob Orfali had a veritable field day creating an aura for distributed Java objects. In 1997, if you weren't engaged in IDL, RMI, CORBA, IIOP and Java, you might as well have been living in a cave.

During the spring of 1997, in the midst of the second Java One conference, JavaSoft announced the Enterprise JavaBeans specification plans. The promise was impressive: server side components that could interoperate and be part of a comprehensive enterprise class processing environment (messaging, transactions, persistence, etc.). Of course, a specification is just that – a spec.

By the following Java One conference, several companies formally announced plans to ship EJB implementations. Two of these companies were Weblogic (Tengah) and Progress Software (Apptivity 3.0).

One of the major significances of EJBs is that its design can bring the powerful concept of server, business logic componentry to the masses. Sure, with RMI or CORBA, business logic components could always be achieved by the gurus. But a really great EJB implementation, coupled with tool support, can allow an "average Java Joe" to make use of the organization, design and infrastructure. That is very significant.

Remember what SQL's "SELECT * FROM CUSTOMER" did for the average programmer faced with database query challenges? An EJB implementation, in and of itself, is not enough. The average developer and the developer with serious productivity requirements demand an integrated environment with EJB components, JavaBeans, an event infrastructure and state of the art IDE to put it all together.

As we stand today, EJBs are already receiving criticism for not being "run anywhere, anytime" – the familiar psalm of the Java-critical choir. The complaint is that an EJB from one EJB container cannot instantly be ported to a different EJB container without significant effort. The fact is that every EJB implementation vendor is adding "special sauce" to its product – this is good business practice and it benefits the developer in the long run.

What if All Database Vendors Provided Only Strict SQL Implementation?

Even client-side JavaBeans didn't easily move from one design time container (IDE) to another. For example, Sun's own Java Studio produced Beans that were wrapped to provide additional value within Studio, but did not port to other IDEs.

This is not such a big problem, however, the actual Bean part of the wrapped component can always be had by the developer or put forward by the product of the Bean. Also, this lower denominator is very adept at transport. The producer of EJBs or JavaBeans is responsible for putting forward a full-blown "special sauce" component and a plain vanilla specification component, whenever possible.

RMI, CORBA, EJB – the movement toward distributed computing and components is on and it's headed for the masses. In 1999 the average Java Joe developer will definitely be able to create and use Enterprise JavaBeans in the construction of Networked Applications. This is a big step forward for Java as well as the vendors providing solutions to developers.

Java George is George Kassabgi, director of developer relations for Progress Software's Apptivity Product Unit. You can e-mail him at george@apptivity.com.

george@apptivity.com

ObjectSpace

www.objectspace.com