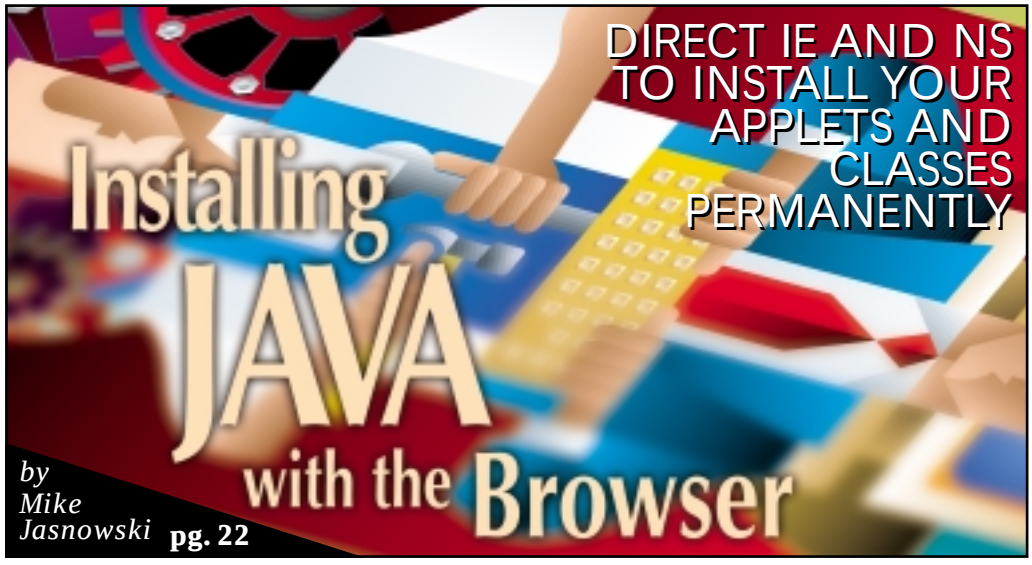
UNLOCKING THE SECRETS OF THE JMF

JAVA™ DEVELOPER'S JOURNAL

The World's Leading Java Resource

Volume: 5 Issue: 3, March 2000

SYS-CON PUBLICATIONS



Hot Dispatch

www.hotdispatch.com

JAVA DEVELOPER'S JOURNAL

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

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

WRITERS IN THIS ISSUE
LINDEN DECARMO, RICK HIGHTOWER, JUSTIN HILL, MIKE JASNOWSKI,
TONY LAPASO, GABOR LIPTAK, BARRY MOSHER, SEAN RHODY,
TIMO SALO, JON SIEGEL, DAVID SKOK, A.V.B. SUBRAHMANYAM,
SAM WATTS, JASON WESTRA, ALAN WILLIAMSON

SUBSCRIPTIONS
FOR SUBSCRIPTIONS AND REQUESTS FOR BULK ORDERS,
PLEASE SEND YOUR LETTERS TO SUBSCRIPTION DEPARTMENT
SUBSCRIPTION HOTLINE: 800 513-7111
COVER PRICE: $4.99/ISSUE
DOMESTIC: $49/YR. (12 ISSUES) CANADA/MEXICO: $69/YR.
OVERSEAS: BASIC SUBSCRIPTION PRICE PLUS AIRMAIL POSTAGE
(U.S. BANKS OR MONEY ORDERS). BACK ISSUES: $12 EACH

PUBLISHER, PRESIDENT AND CEO: FUAT A. KIRCAALI
VICE PRESIDENT, PRODUCTION: JIM MORGAN
VICE PRESIDENT, MARKETING: CARMEN GONZALEZ
ACCOUNTING MANAGER: ELI HOROWITZ
CIRCULATION MANAGER: MARY ANN MCBRIDE
ADVERTISING ACCOUNT MANAGERS: ROBYN FORMA
MEGAN RING
JDJSTORE.COM: JACLYN REDMOND
ADVERTISING ASSISTANT: CHRISTINE RUSSELL
GRAPHIC DESIGNERS:: JASON KREMKAU
ABRAHAM ABBO
GRAPHIC DESIGN INTERN: AARATHI VENKATARAMAN
WEBMASTER: ROBERT DIAMOND
WEB SERVICES CONSULTANT: BRUNO Y. DECAUDIN
WEB SERVICES INTERN: DIGANT B. DAVE
CUSTOMER SERVICE MANAGER: CAROL KILDUFF
CUSTOMER SERVICE: ANN MARIE MILILLO
ONLINE CUSTOMER SERVICE: AMANDA MOSKOWITZ

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

JAVA DEVELOPER'S JOURNAL (ISSN#1087-6944)
is published monthly (12 times a year) for $49.00 by
SYS-CON Publications, Inc., 39 E. Central Ave., Pearl River, NY 10965-2306.
Periodicals Postage rates are paid at
Pearl River, NY 10965 and additional mailing offices.
POSTMASTER: Send address changes to:
JAVA DEVELOPER'S JOURNAL, SYS-CON Publications, Inc.,
39 E. Central Ave., Pearl River, NY 10965-2306.

WORLDWIDE DISTRIBUTION BY
CURTIS CIRCULATION COMPANY
739 RIVER ROAD, NEW MILFORD NJ 07646-3048 PHONE: 201 634-7400

BPA INTERNATIONAL
SYS-CON PUBLICATIONS

SEAN RHODY, EDITOR-IN-CHIEF

Going, Going, GONE!

I may be somewhat unusual, but I've never bought anything at an online auction. I've seen eBay, and one of my friends sold some of his collection of valuable magazines (okay, comic books) on eBay, but I've never gone the whole route and come home with the goods. I've thought about it a couple of times. I recently built a computer from parts, and one of the places I looked for motherboards had an auction, but they didn't have the board I wanted and there was no way to post my own offer to buy.

Auctions consist of at least two different processes, if not three. There's the straightforward auction, in which a seller posts an offer of something that multiple buyers can bid on. Then there's the reverse auction, in which a buyer posts an offer to buy something, and multiple sellers bid progressively lower prices (at least in theory). And there's the true exchange, in which multiple buyers and sellers gather and a product changes hands whenever two or more participants can agree on a price for particular goods.

Of course, there are multiple variations on auctions. Things like English auctions, Dutch auctions, silent bids, and so forth. These all have more to do with the behavior of the auction than with the concept of the auction itself. Rules such as minimum bid intervals, who can see information, who can participate are all part of the package, but the concept stays the same: buy and sell.

A number of vendors produce software to implement auctions and exchanges, including companies like Trading Dynamics, Tradex and Tradium. You might have heard about Ariba buying both Trading Dynamics and Tradex. This industry is hot.

But it's also off target, at least slightly. I work in Net Markets, and meet with clients daily. I wish I had stock options for every time I've gone into a conversation with a potential new market that thinks they want auctions. They don't. Not really – or at least not only.

This is true for several reasons. Now, most of my conversations have been with business-to-business (B2B) clients rather than consumer-to-business (C2B) clients. On the consumer side, the auction model has more validity, because in general you're selling products with a relatively small price tag in relatively high quantity. But B2B is different. A consumer site like eBay may have millions of participants, but many business-to-business sites have audiences in the hundreds because they focus on either high unit price or high-volume orders. Only so many companies want to trade in airplane parts, for example.

That's not to say this isn't a good market to be in – a book may cost $5.99 on Amazon.com, so their model is clearly volume: making millions by moving a large number of small-ticket products. An airplane engine may be worth millions, so the number of deals will be smaller, but the overall dollar volume may in fact be much larger than most C2B sites.

And that's one of the things that drives B2B to need more than auctions. Remember, an auction is essentially an open market where the only relevant attribute of the market is price. Exchanges are slightly more complex, but the main element of an exchange is still price matching between the buyer and seller.

That's where the model breaks down in B2B. Most B2B sites aren't set up as a buyer or a seller, but as a broker. They take a commission on the sale – most times from the seller, sometimes from the buyer, occasionally from both. But because of the large price structure, and the nature of their products, an auction isn't always appropriate.

Most high-dollar products have multiple properties that affect the trade in different ways. For example, a product may come in several grades. An offer to buy might be for the higher grade, but in reality the buyer might be able to use the next highest grade should other options such as price make it attractive. So the participants in this kind of market would want to trade not only on price but also on grade. Then shipping, or location, or packaging or a hundred other factors may come into play. Technically, they want multiple attribute trades. And what they really want is negotiations.

An auction almost always has an automated clearing. The highest bidder wins when the auction ends. A negotiation doesn't necessarily end that way, because price is only one criterion for assessing the best deal. You might have to figure in freight charges, which may be lower if the seller is in the same state or higher if the seller is out of the country. You might be comparing two similar grades, or two grades that can do the same job but have different efficiencies that need to be accounted for. The rules are generally too complex to code to an automatic conclusion.

Other factors come into play. Auctions and exchanges are designed to be open to as wide an audience as possible because it drives liquidity. Business deals don't always work that way. A company that sells steel to General Motors for $80 a ton doesn't necessarily want GM to be able to bid on their steel at $60 a ton in an online market, or even to know that they are providing steel at that price. Anonymity is usually a requirement, but restriction of trading partners is also common. These tend to constrain a market somewhat, but since many of these markets are for multiple-attribute products, a commodity approach may not be appropriate anyway.

Unfortunately, at the moment there's a big, negotiation-engine–shaped hole in the market. There are no shipping products that handle negotiations; most really can't even deal with multiple attributes. The race to get there first is heating up; Ariba isn't the only company buying market product makers. Better get in the bidding fast for that first negotiation company, because it'll quickly be going, going, *gone!*

sean@sys-con.com

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

WRITTEN BY DAVID SKOK, CHAIRMAN/FOUNDER OF SILVERSTREAM SOFTWARE

Battle in the Making

Portals, e-commerce sites, B2B commerce and so on…we're witnessing unprecedented demand for e-business solutions of every stripe as companies rush to put their businesses on the Web. With Y2K now out of the way, this has become the top IT priority.

In the past many of these solutions have been built on top of IRM (Internet Relationship Management) products such as Vignette and BroadVision whose primary specialization has been content management and personalization. For early Web sites, mostly serving up static content, this was sufficient. But once the basic site is up and running, demand quickly shifts to more data-driven applications in which customers can place orders, see inventory, book travel, track packages and the like. This requires the ability to develop highly scalable, enterprise-class Web applications that can access existing corporate data and applications and provide high-end transactional capabilities. This kind of capability is available only from application server and solutions vendors.

Because of the demand for both content management/personalization and back-end data access/transactions, we're about to see a collision between these two markets. In particular, almost every application server vendor has begun developing e-business solutions that include IRM functionality based on top of their application servers to meet the demand from their clients for more prebuilt functionality.

An important backdrop to this upcoming battle is the advent of an extremely important industry standard, J2EE (Java 2 Platform, Enterprise Edition), that standardizes the way application servers should work. This standard has been uniformly adopted by almost every application server vendor, and is being enthusiastically adopted as the standard for Web development by much of corporate America.

J2EE is creating tidal waves in the Web application marketplace just as SQL did when it swamped all other relational databases.

Prior to the standard, Web software companies – including IRM vendors such as Vignette and BroadVision – wrote their own servers to support their applications and frequently spent as much as 70% of their energy on doing so. These were proprietary, with weak architectures missing important features such as distributed objects, transactions, standardized data access and message queuing. Furthermore, they provided proprietary scripting languages that weren't up to the task of writing serious enterprise applications. Today no software company in its right mind would write its own proprietary server. It would simply write to the J2EE standard and support the leading J2EE servers in the market.

Because of the availability of J2EE standardized servers, there's a backlash at customer sites against the proprietary server architectures. Organizations standardizing on J2EE want to use this architecture for all their e-business applications. That way they invest in learning a single skillset that can be used across all projects, and they're easily able to find developers who already know the J2EE standard. They're also able to use a much wider range of development tools since products such as Rational Rose and Macromedia DreamWeaver are adapted to support the J2EE standard. In addition, they can purchase add-on components and prebuilt software modules from third-party vendors that comply with the J2EE standard.

As the content management/personalization vendors collide with the application server vendors, you can see a major battle brewing. At stake is the huge e-business platform market. In the one corner are Vignette, BroadVision and similar products, built on a proprietary architecture with weak scripting capabilities. In the other corner are the application server vendors who are moving into e-business solutions, leveraging their strong architecture.

The obvious move for the existing IRM vendors is to rewrite their products to be J2EE compliant. Companies such as Vignette are describing multiphase plans to do something like this. But it's a difficult task that may well take years to accomplish correctly, during which time it may be close to impossible to simultaneously evolve their solutions functionality at a rapid pace. Probable outcome: loss of momentum and marketshare.

Several application server vendors are already pursuing the strategy of adding e-business solutions to their product line and moving into the complete e-business platform space. Not all will be successful, since the *solutions* skillset differs from the *systems* skillset that most possess. This sets the stage for a series of important new battles: it'll shortly become clear that an application server without an IRM layer offering higher-level solutions functionality such as content management, personalization, user profiling and shopping baskets will no longer be considered adequate.

Buyers are placing more and more emphasis on this part of the vendor's offering, and view the J2EE application server as a standardized component that isn't highly differentiated from one player to the next. So the winners will be those who best understand the requirements in this solutions area and who move fastest to create the appropriate product offering.

Ultimately, an interesting battle will start to evolve between the traditional IRM vendors and the new application server-based IRM offerings. As the application server vendors reach critical mass with their functionality, we'll see them begin to take significant market share away from the traditional vendors because their industry-standard application server-based solutions will be more complete and will offer far greater flexibility and customizability.

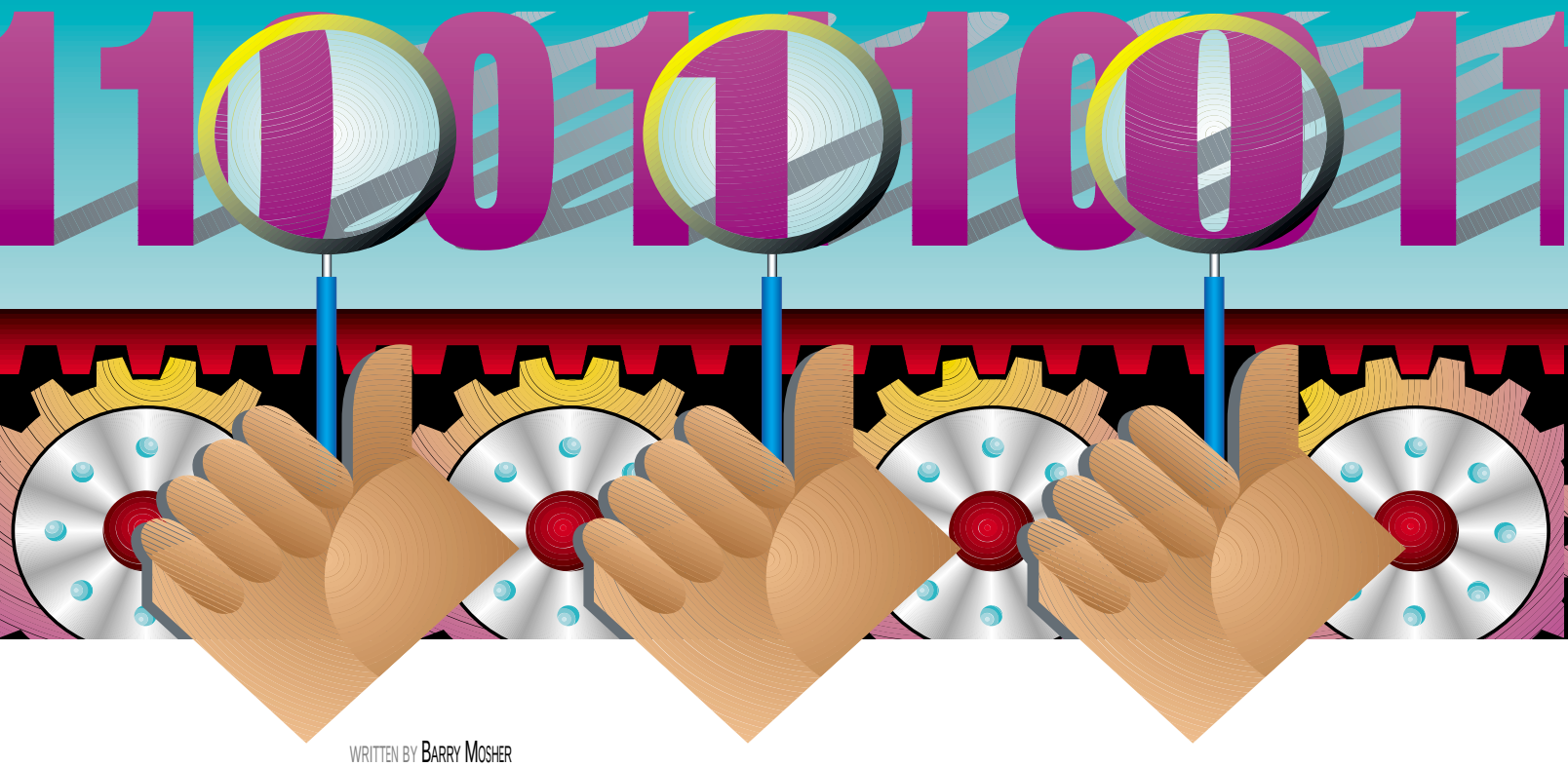
dskok@silverstream.com

AUTHOR BIO

David Skok holds a degree in computer science from the University of Sussex in England and has started up successful companies in England, Germany and France. He founded his first company (CAD/CAM) at age 22.

USE EXCEPTION CHAINING

HOW TO RECORD THE ROOT CAUSE OF HIGH-LEVEL



WRITTEN BY BARRY MOSHER

Exception chaining (also known as "nesting exceptions"), is a technique for handling exceptions. A list is built of all the exceptions thrown as a result of a single originating exception as it's converted from lower to higher levels of abstraction. It can be used in both client and server environments to greatly simplify software debugging without adding undue complexity. This article discusses good exception-handling techniques and shows how to implement and use exception chaining.

Exceptions Are Part of the Interface

When our professors taught us about encapsulation and modular programming, we were told that modules should hide the details of their implementation. For example, an employee lookup service shouldn't advertise how or where it finds employees; only the methods required to access this service should be publicly available. This reduces complexity by isolating the implementation of one component from others that use it.

Most programmers realize that exceptions form an important part of the interface of any class, but aren't sure what to do with low-level exceptions that can't be handled directly by their code. Throwing a low-level exception from a higher-level class isn't a good idea because it exposes details of that class's implementation. The correct solution is for the class to catch the low-level exceptions, and to rethrow exceptions of a higher level of abstraction. If the getEmployee() method retrieves employee objects from a database via JDBC, for instance, a SQLException might be caught inside the method. This exception would be converted to an EmployeeLookupException and rethrown to the calling method.

Converting an Exception Object

There's no way, of course, to simply change the type of an exception. While it might be argued that we're converting from a more to a less specific exception, we can't cast one to the other because they don't have a parent–child relationship. For example, SQLException certainly doesn't extend from EmployeeLookupException or vice versa. In this article "converting" an exception means to catch one exception type and throw a brand new exception of a different class. It's a conversion in the sense that the new exception is a direct result of the earlier one. When the EmployeeLookupException is created and thrown, it doesn't mean that a second error has occurred; it's just new packaging to represent the original error at a higher level of abstraction.

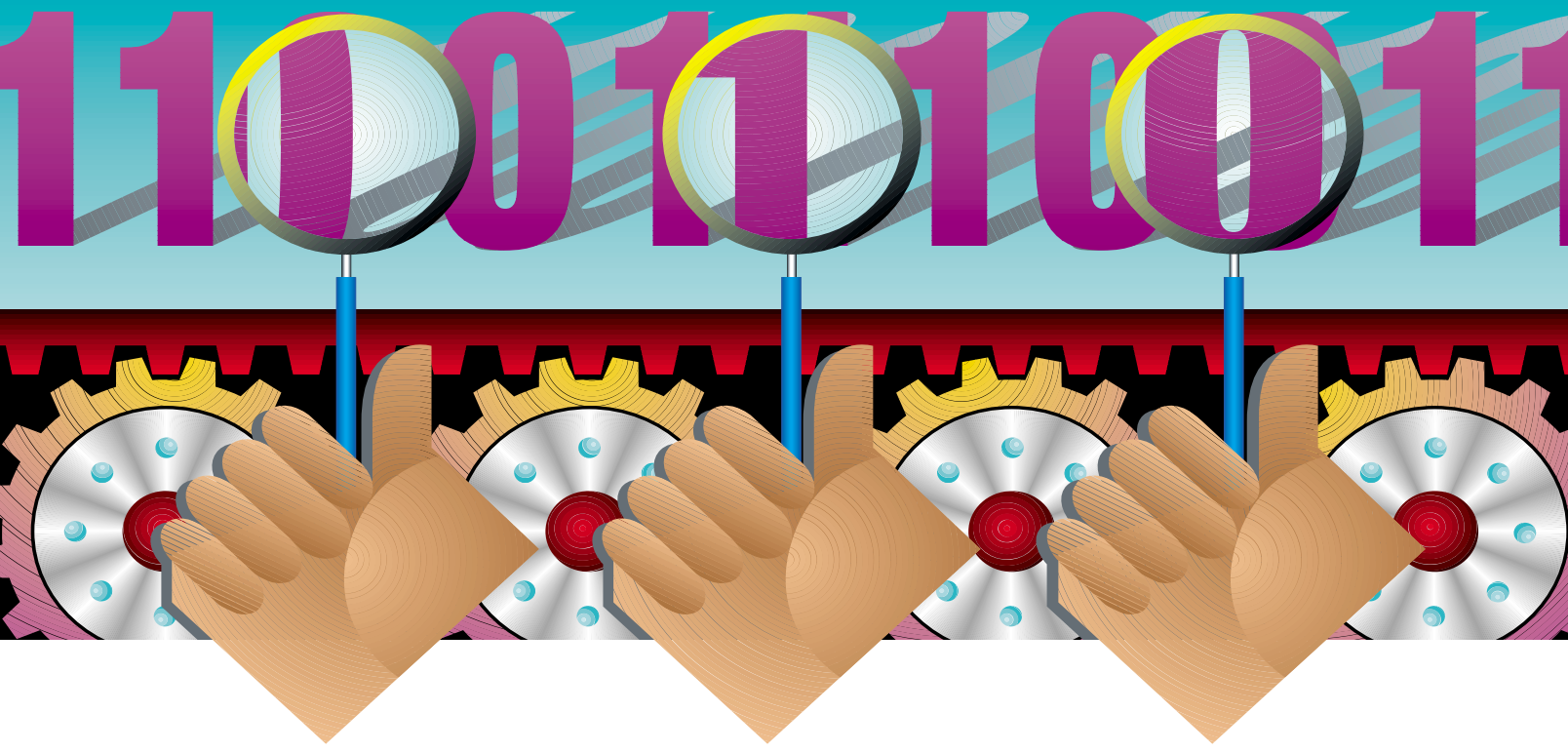
```
try {
   stmt.execute(sql);
} catch (SQLException ex) {
   throw new
      EmployeeLookupException();
}
```

It quickly becomes obvious to many programmers that exception conversion (without chaining) has a serious deficiency: the root cause of the exception is lost. Is the EmployeeLookupException a result of a logon failure or a SQL query error? Exception conversion can make debugging more difficult.

I've argued in the paragraphs above that details of an implementation should be hidden from the user of the method, but we don't want to make debugging more difficult than necessary. The distinction to be

TO SIMPLIFY DEBUGGING

EXCEPTIONS WITH EXCEPTION CHAINING



made is that the internal implementation details should be hidden from the public API used at compile time, but this isn't a reason to obscure the cause and location of internal exceptions at runtime. We want to make these easy to discern. Correct exception handling with exception chaining achieves both these goals.

Upcasting and Other Poor Exception-Handling Techniques

One common solution to the problem of losing information with exception conversion is to convert through upcasting (i.e., casting "up" the tree to a superclass). Usually, the up-cast is to Exception and is programmed in an implicit manner by simply declaring that each method throws Exception. This undermines the intent of requiring exceptions to be caught in Java. The compiler doesn't require the programmer to catch *any* exceptions by this method.

```
public Employee getEmployee()
   throws Exception
{
   ... database query code ...
}
```

While proper exception conversion adds context, making it more specific to the problem at hand, upcasting makes it more difficult to determine how to handle the exceptions. The desired exception conversions are from lower to higher levels of abstraction. This rarely matches the inheritance hierarchy of exceptions (which changes in amount of detail rather than abstraction – a very subtle difference). Remember that the abstraction a programmer seeks in designing a class should relate to the problem area he or she is trying to solve. For example, Exception is more generic than SQLException (a subclass of Exception), but it's probably not a higher level of abstraction for an employee lookup service. EmployeeLookupException and NoPermissionException are good examples of a higher level of abstraction in this case.

An even worse strategy for exception handling is to catch and log each exception (or not) before ignoring it. As long as the exception is logged, debugging is relatively straightforward. Unfortunately, this leaves no way for the calling method to detect or handle the error, which again eliminates the value of having exceptions.

What Is an Exception Chain?

An exception chain is a list of all the exceptions generated in response to a single root exception (say, a SQLException). As each exception is caught and converted to a higher-level exception for rethrowing, it's added to the chain. This provides a complete record of how an exception is handled (see Figure 1).

```
try {
   stmt.executeUpdate(sql);
} catch (SQLException ex) {
   throw new
      EmployeeLookupException(
         "Query failure",ex);
}
```

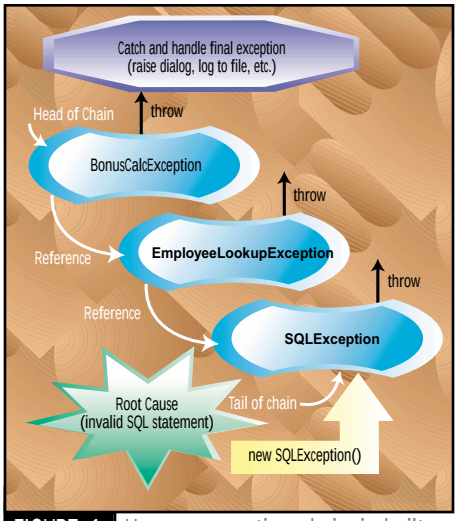


FIGURE 1 How an exception chain is built

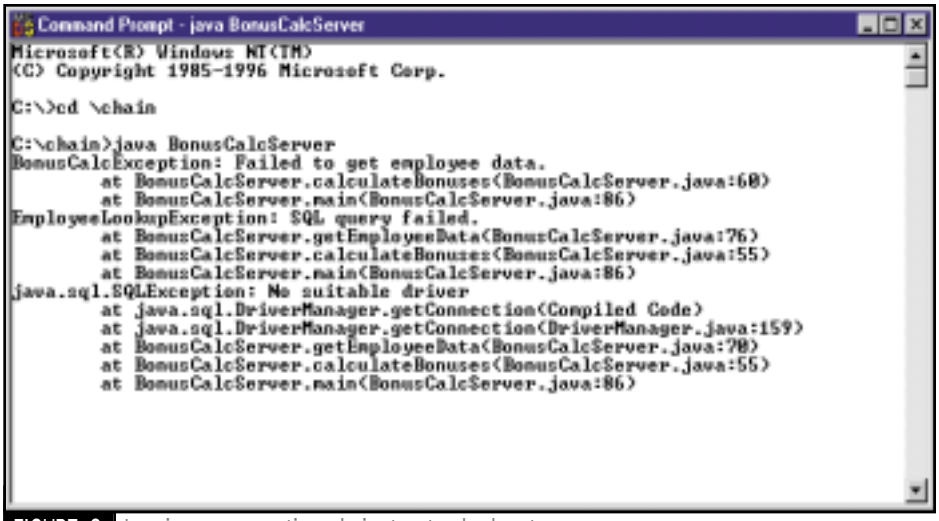


FIGURE 2 Logging an exception chain to standard out

The Implementation

The exception chain is implemented as a linked list of exceptions in reverse order from last exception thrown to first. Each exception is a link in the chain. The first exception thrown that begins the chain can be of any type – there's no need for any special functionality – but all subsequent exceptions must have a way of referring to the previous exception so that the chain of exceptions can be maintained. In addition, there must be some means for accessing/examining the exception chain. I refer to exceptions that provide these things as being "chainable" – that is, an exception chain can be constructed using them.

For my purposes a "chainable" exception must do two things:

1. Provide at least one constructor taking another (previous) exception as a parameter and storing it.
2. Override each of the printStackTrace() methods to first print their own stack trace, then invoke the corresponding printStackTrace() method of the previous exception (given as a parameter in the constructor).

The first requirement enables the exception chain to be built. The second provides a simple means of displaying the results by invoking printStackTrace().

In practice, these simple rules allow chaining to be added to exception classes without forcing undue requirements on the users of those classes. A programmer doesn't need to know about exception chaining, even when using chainable exceptions. If chaining is used (by passing the previous exception to the new exception when converting), then the benefits are seen in the stack traces. If it's not used, there's no learning curve or overhead for the developer, and he or she is no worse off than if using nonchainable exceptions.

Any exception class can be written to meet the requirements described above, regardless of what its superclass is.

Let's examine a simple implementation of a chainable exception that extends Exception. The ChainedException (see Listing 1) can be used as a template when writing other chainable exceptions (e.g., copy the code overriding the printStackTrace() methods), or simply extended by those exceptions that would normally extend the Exception class.

Notice that the constructors take Throwables as parameters rather than Exceptions. This allows the "exception" chain to be made up of any classes that can be thrown in Java, including both compile time and runtime exceptions as well as Errors. The implementation of these constructors is straightforward; they save the reference to the given Throwable instance with the _previousThrowable member variable. This reference is used only by the printStackTrace() methods.

Also note that some constructors don't take any Exceptions or Throwables as a parameter. These constructors are used to create the root exception.

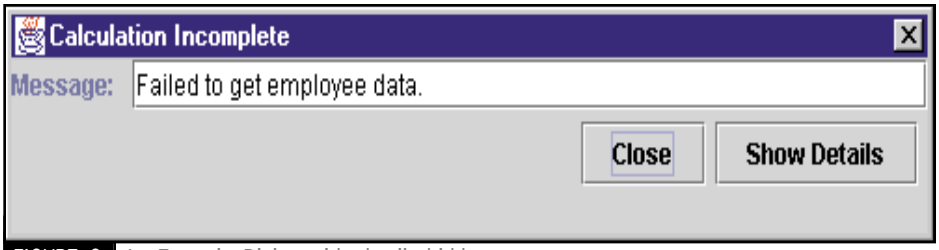


FIGURE 3 An ExceptionDialog with details hidden

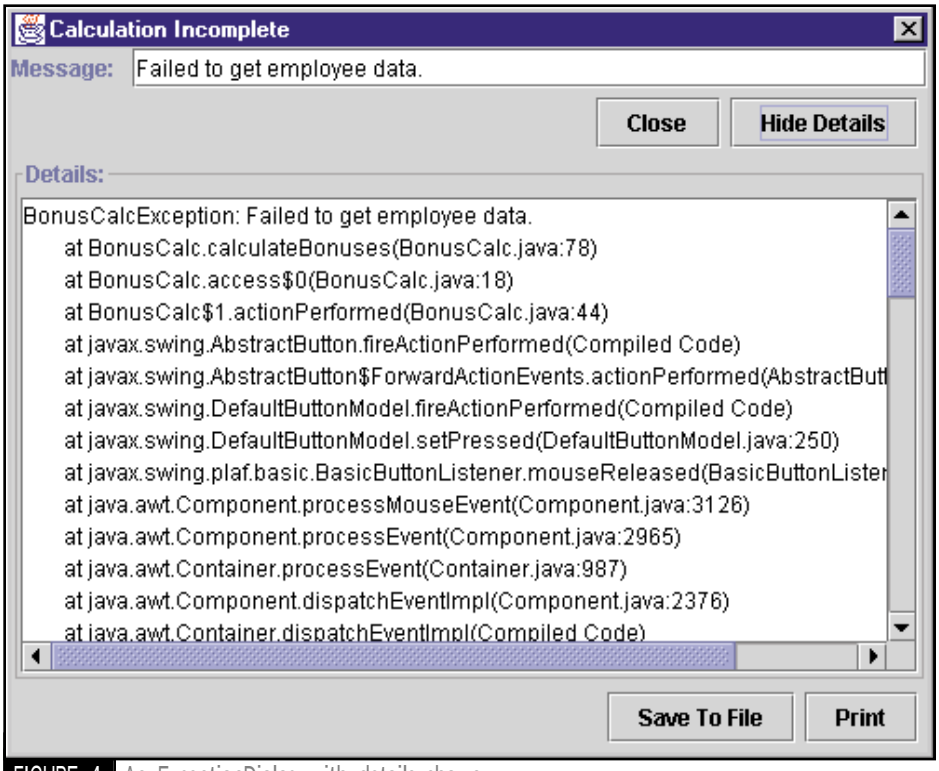


FIGURE 4 An ExceptionDialog with details shown

Server (and Non-GUI) Applications

Using exception chaining in server applications is simple. Build the chain as normal, and when logging exceptions be sure to use one of the printStackTrace() methods. There's no need to log each exception as they're caught. Only the last exception of the chain has to be logged, since it will display the message and call stack of each exception in the chain.

There's no harm if intermediate exceptions are logged individually, however; the worst that can happen is a slightly more cluttered log (see Figure 2). Server administrators are familiar with reading logs to determine errors, and are grateful for the extra information provided by using exception chaining.

The next section describes how a specialized dialog class can make exception chaining accessible in client programs.

Client Applications

One enormous benefit of the Java language is its resilience to errors. Java programs aren't generally "auto-terminated" as a result of an error; instead, the error is thrown (in the form of an exception) up the call stack to be handled. In a Java client application, if a runtime exception isn't handled, the main event dispatcher will print the exception's stack trace to standard out (or a Java console). This simple action is actually quite useful. While the user's button click won't appear to have done anything, at least a record of the error is available (if the Java console is visible and the user knows to look at it). Therefore, many programmers decide to handle all exceptions this way: log the exception and abort the action.

A better approach is to inform the user of any errors, usually via a dialog box. The best dialogs clearly indicate two things: what action failed, and why. I've implemented a dialog box (called ExceptionDialog) for use with exceptions that provide that information. It can be used equally with exception chains and single exceptions; it doesn't distinguish between the two.

To use the ExceptionDialog, a "try/catch" block is added to the individual GUI event handlers the programmer has registered. When an exception is caught by an event handler, an instance of ExceptionDialog is created (passing the exception as a parameter) and displayed. The title of the dialog (also passed in as a parameter to the constructor) indicates to the user what requested action has failed to occur because of the exception being thrown. The dialog message string is populated with the result of calling the exception's getLocalizedMessage() method, providing the second important piece of information – why that action couldn't be completed. Figure 3 shows an ExceptionDialog with details hidden.

```
try {
   ... event handler code ...
} catch (Exception ex) {
   ExceptionDialog dlg =
      new ExceptionDialog(
         frame,ex,
         "Look-up failed.");
   dlg.show();
}
```

To examine the cause of the exception in more detail, the user clicks the "Show Details" button to expand the dialog. The expanded dialog (see Figure 4) displays the call stack of the exception and, if it's a chain of exceptions, the call stack of each exception in the chain. The information can be printed, or copied into an e-mail and sent to the user's help desk.

Listing 2 demonstrates exception chaining and use of the ExceptionDialog class in a GUI program. The BonusCalcException and EmployeeLookupException classes (not shown) both inherit from ChainedException.

The complete source code for all classes can be downloaded from www.JavaDevelopersJournal.com.

Where Have I Seen This Before?

The RemoteException in RMI and the ServletException from the Java Servlets API are both chainable classes. Sun refers to exceptions as being "nested" rather than "chained," but the intentions are the same.

Working on the Chain Gang

My own experience with exception chaining has shown it to be very valuable, and I tend to use it in most projects I'm involved with. However, I've found one common scenario that presents new hazards: distributed Java programs using RMI (remote method invocation).

The benefit of using RMI is that the distinction between local and remote objects is greatly reduced. It's not much more difficult to invoke a method on a remote object than a local object, except that the calling method must catch or throw RemoteException. Using exception chaining with exceptions that may be thrown during a remote method invocation offers the same benefits as with regular method invocations. It's easy to determine the root cause of errors even if they occurred in a remote object.

Unfortunately, while the sets of classes available to client and server applications overlap, they're not usually identical. Server programs have no need for GUI classes, and client programs have no need for database classes. The set of classes (or at least JAR files) available to the client is often intentionally stripped down to the minimum number required for its functionality to minimize its footprint. Therefore, if an exception chain containing a database-specific exception is thrown from the server to a remote client, it may not be possible to deal with it because the client application doesn't know what that database exception is. A ClassNotFoundException will be thrown when there's an attempt to unmarshal the unknown exception class.

Exception chaining can still be valuable in distributed applications. The easiest way to prevent the problem described above is for each application to be responsible for reporting its own exceptions. Methods that can be invoked remotely should log their internal exceptions and create a new exception for throwing remotely. The new exception should not chain the previous exception. This guarantees that no unknown exception types will be passed remotely by an exception chain.

Summary

Exception handling is often seen as a hindrance and treated as an afterthought. In fact, good exception-handling techniques can greatly simplify debugging. The exception-chaining technique is useful for both client and server components, making it easier to determine the root cause of exceptions without increasing the complexity of work required to handle them.

Author Bio

Barry Mosher is a Java consultant in the San Francisco Bay Area. A Sun-certified Java programmer, Barry has more than seven years of programming experience, including three in Java.

barrymosher@netscape.net

Listing 1: The ChainedException Class

```
// Filename:  ChainedException.java
// Author:    Barry Mosher

import java.io.PrintStream;
import java.io.PrintWriter;

public class ChainedException extends Exception
{
   private Throwable _previousThrowable = null;

   public ChainedException() {}

   public ChainedException(String pMsg) {
      super(pMsg);
   }

   public ChainedException(Throwable pEx) {
      _previousThrowable = pEx;
   }

   public ChainedException(String pMsg,
                           Throwable pEx) {
      super(pMsg);
      _previousThrowable = pEx;
   }

   public void printStackTrace() {
      super.printStackTrace();
      if (_previousThrowable != null) {
         _previousThrowable.printStackTrace();
      }
   }

   public void printStackTrace(PrintStream pPS) {
      super.printStackTrace(pPS);
      if (_previousThrowable != null) {
         _previousThrowable.printStackTrace(pPS);
      }
   }

   public void printStackTrace(PrintWriter pPW) {
      super.printStackTrace(pPW);
      if (_previousThrowable != null) {
         _previousThrowable.printStackTrace(pPW);
      }
   }
}
```

Listing 2: Example Use of Exception Chaining

```
private void registerListeners()
{
   calcBtn.addActionListener(new ActionListener()
   {
      public void actionPerformed(ActionEvent e)
      {
         try {
            calculateBonuses();
         } catch (Throwable ex) {
            // Handle all exceptions
            // by displaying a dialog.
            ExceptionDialog exceptionDialog =
               new ExceptionDialog(
                  BonusCalc.this,ex,
                  "Calculation Incomplete",true);
            exceptionDialog.show();
         }
      }
   });
}

// Simulates a calculation routine that can
// fail and throw exceptions.
private void calculateBonuses()
   throws BonusCalcException {
   try {
      getEmployeeData();
   } catch (EmployeeLookupException ex) {
      // Convert the exception to a higher
      // level of abstraction suitable for this
      // method.
      throw new BonusCalcException(
         "Failed to get employee data.",ex);
   }
}

// Simulates a query routine that can
// fail and throw exceptions.
private void getEmployeeData()
   throws EmployeeLookupException {
   try {
      Connection con =
         DriverManager.getConnection(
         "invalid_connection");
   } catch (SQLException ex) {
      // Convert the exception to a higher
      // level of abstraction suitable for this
      // method.
      throw new EmployeeLookupException(
         "SQL query failed.",ex);
   }
}
```

Couldn't String Two Words Together

When I sat down to write this month's column I tried desperately to come up with something. I was beginning to panic, as nothing seemed to come to mind. Then I took a wee walk and munched down on some pizza. Suddenly, *BANG!* The whole month's activities came flooding back to me. So kick back, take some time out and lend me your eyes for, oh, I don't know, 10 minutes.

WRITTEN BY
ALAN WILLIAMSON

Now that I think back, it was a fun-filled month with huge amounts of goings-on. If you remember last month's column you'll know that, of late, I haven't been very impressed with what's happening at Sun. I essentially hinted at the fact that we ought to keep an eye on Scott McNealy. Ironically, this month I discovered a wee piece of information that makes my concerns even more relevant.

Will Sun Eclipse Linux?

Ask anyone which operating system is capturing the headlines. Unless they've been hidden away in the darkest corners of the Redmond campus, the answer will usually be Linux. Why? One of the main reasons is its cost – the damn thing is free. Another reason is performance – it works extremely well with very little hardware. Couple these two excellent advantages and you lock out most of the competition.

But what if the competition gave away *their* operating system? That would definitely weaken the justification to move to Linux. Well, guess what our friends at Sun have just gone and done? Beginning this month, Solaris 8 will be free for both Sun and Intel platforms. Quite a bold move when you think about it, and one that I believe will hit Linux very hard.

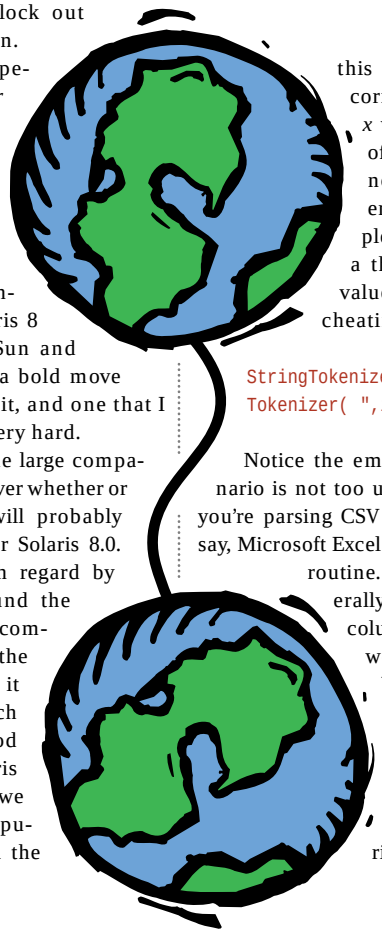
I suspect many of the large companies that are dithering over whether or not to install Linux will probably hang back and wait for Solaris 8.0. Solaris is held in high regard by many developers around the planet, and I know my company will be installing the beast and measuring it against Linux. So watch this space. One good thing: now that Solaris will be freely available, we won't have to wait for popular ports to Linux from the large vendors that have already developed Solaris ports. I'm interested in watching this one and seeing where it heads. Let me know what you think through our discussion list (details of which are given later).

StringTokenizer

This month I've been wrestling with the core class java.utils.StringTokenizer, and spent a bit of time trying to get it working the way the documentation suggests. (Anyone on our mailing list will know this story, so if that includes you, skip past and move on to the next section. I won't be offended, honest.)

My problems started when I wanted to spot fields with empty tokens. Let's consider the following, for example:

```
StringTokenizer ST = new StringTok-
enizer( "this,is,a,test", "," );
int x = ST.countTokens();
```



A standard use of this class, and you'd be correct in assuming that *x* would be 4. This use of StringTokenizer has no problems. Consider the following example, however, and have a think about what the value of *x* would be. No cheating or reading ahead!

```
StringTokenizer ST = new String-
Tokenizer( ",is,,test", "," );
```

Notice the empty tokens. This scenario is not too untypical, especially if you're parsing CSV files produced from, say, Microsoft Excel or some other export routine. An empty token generally denotes an empty column, so it's a scenario we need to know about. Well, sadly, StringTokenizer returns back 2 in this instance, which poses a potential problem for any parsing algorithm.

Another problem with this class is its inability to cope with nonstandard delimiters. Try parsing a string that's delimited by, say, the ">>" sequence. It won't work. However, if you follow the documentation, the constructor for StringTokenizer accepts a String instance for the delimiter and nowhere suggests that you can't use any nonstandard delimiter. If indeed this class is meant to use standard delimiters, the constructor should only be accepting a flag or something that forces the developer to use it. For example:

```
StringTokenizer ST = new StringTok-
enizer( "this,is,a,test", StringTok-
enizer.COMMA );
```

...which would make far more sense than what happens now. At least this takes the guesswork out of it. We discussed the problem at great length on the mailing list, and I believe it was Blair Wyman who highlighted the fact that strtok in C operates along the same lines. It could be argued that StringTokenizer merely provides an easy transition path, but I don't accept that. This is Java, not C. Java has an opportunity to get all the quirks ironed out once and for all, presenting the developer with a clear, concise library of tools.

I submitted this to the official bug report, and asked that they either change the functionality of StringTokenizer or update the documentation to warn developers that it doesn't do what you might expect it to. As part of the bug report, I sent an alternative implementation to StringTokenizer, which we use as part of our n-ary core library set. Ours works. If you want to vote for this bug, or check out its latest status, the bug ID is 4133287. The official response from Sun on a number of other StringTokenizer-related bugs is: "It's not meant to be a fully functional string scanner." What sort of answer *that* is, I have no idea. I'll keep you updated on this one.

Support Award of the Month: Hall of Shame

This month we recruited a couple more developers to our team here in Scotland. This meant that we had to purchase a couple more development machines for them to use. We usually purchase from either Gateway or Dell, depending on who's offering the most for the lowest price. Nine out of 10 times there's little difference between the two, and this time we plunked for Dell. What a painful choice this turned out to be!

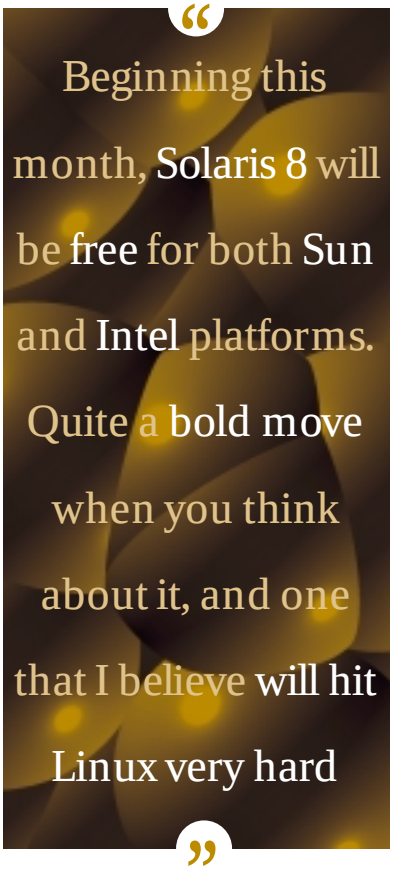
As some of you know, I'm no fan of Microsoft's ASP technology, which I've dubbed "A Severe Pain!" – I prefer the much more portable Java Servlet solution. Anyone who actually advocates *any* Microsoft solution for a high-traffic Web site requires a complete brain-bios update. Dell is a great example of a site that should really rethink its back-end technology, and do it fast. To cut a long story short, I set out to order two machines – and ended up getting six!

The Web order failed right at the end where you submit your confirmation of the transaction. A wonderful "ASP scripting error" page comes up. Okay, not a problem, methinks. I'll report it to them so someone can fix it. The least I can do. I quickly locate the Customer Services section and proceed to fill out the form indicating my problems, only to have another ASP error when I hit the submit button. *Argh*! The *whole site* is based on ASP! Disaster. I spent a long time looking for a page with normal e-mail and after 20 minutes I found one. But I'm still waiting for a response. Having ventured unsuccessfully into the world of e-commerce, I reached for the tried and trusted phone and placed my order with a sales representative. You ring up Dell and a phone menu is read out to you. If you press Business Sales, you're put through to someone instantly. But, oh, boy, it's a different story when you want to speak to Customer Services!

Dell is being a little too clever with their database, and before I knew it I had ordered the machines for the company that *used* to be in the building, with the actual machines being shipped to our old (three years old) London address, 400 miles away. Go figure! The fax confirmation indicated that you should simply ring up your sales representative and they'll make any corrections. Oh, if only it were that simple!

We phone, and after 45 minutes we get through to someone new. They say they can take the corrections and e-mail them to our sales representative. Eh??? Oh, well, who am I to criticize Dell's well-oiled engine. Another fax confirmation comes through, but this time it's for four machines, but two of them are being delivered to our new address, so I guess it's an improvement.

We ring again. Wait for about the same amount of time – and get through to yet another lady, who takes around 25 minutes to fix our problem. Remember, it took only 10 minutes to place the order in the first place! She said she couldn't update the original order as only our sales representative had access to that. She said she'd e-mail him with the changes. Oh, here we go again.



I said, "Can you cc me into that e-mail?" The answer, I think, will go to the grave with me. She said that this wasn't possible as they worked on an internal e-mail system that wasn't available on the open Net. Dell, one of the largest companies in the world, not able to send e-mail? I think this was a wee fib to discourage customers from e-mailing their sales representatives directly.

Our sales representative eventually called back – four days after the original order was placed – and got it all fixed, but we'll never purchase from Dell again. As regular readers know, I warned last month that I'd be starting this feature, and I'm proud to nominate Dell for this month's Hall of Support Shame.

A classic case of being too quick to get the sale and not really worrying about the actual delivery. This award scheme is out to expose some of these companies and improve the overall customer service we Java people have to put up with.

Mailing List

I think most of you are now aware of the new address for the mailing list server. But I'm still getting requests for the old system, so I guess a lot of you keep your ***JDJ***'s and read them over again. Which is good. The topics of conversation are varied and entertaining. We talk about anything that comes into the mind of our developers. So stop by and join in the fun. To sign up, or even to just stop by and have a look at what's being posted, head on over to http://listserv.n-ary.com/mailman/listinfo/straight_talking.

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

Salute of the Month

This month I'd like to acknowledge my radio cohost, Keith Douglas. This poor man has had, I think, the most stressful month in his life and he has coped admirably. We have had to deal with a particularly awkward project manager who, to be frank, really hasn't a clue about anything technical. Which makes it even more ironic when that person has been placed in a position of responsibility for a software project. I'm sure you've had to put up with similar people. You know the sort, the ones who have no real idea what the cc field in an e-mail is all about and manage to cc everyone and their dog when something goes wrong. Just a shame. The actual fault was on their end, not ours, so lots of red faces. But Keith dealt with her very well and I thank him.

• • •

The time has come for me to bid you farewell. My wonderful Java-touting girlfriend expressed concern over my recent music tastes ranging from Neil Diamond right through to Dolly Parton. Well, the good news is that it's not quite as bad as it used to be. I'm now moving to more mainstream pop, with Geri Halliwell providing this month's entertainment. So with that I say, "Look at Me," and am off!

alan@sys-con.com

ware Spread

que.com

Installing JAVA with the Browser

WRITTEN BY MIKE JASNOWSKI

The use of Java in Web browsers has had mixed results. Applications that run in browsers rather than locally find a host of different hurdles. They're more restricted, run slower at times and take a long time to load, thus making complex applications more difficult. Advances in security and virtual machine technology have addressed the first two items. The third item remains somewhat challenging. Faster modems, increased bandwidth and compressed file formats alleviate the problem somewhat but their impact varies. When fourth-generation browsers appeared, they included some new technology with features that allowed developers to have their applets and supporting classes installed permanently.

This article is focused on distributing Java classes that are permanently installed on a user's machine (see Figure 1). The next time a user visits the page, the classes are loaded locally instead of from the network. I'll start with a short explanation of some concepts and tools used in this process, and apply this knowledge to a working example for Internet Explorer and Netscape Communicator. We'll then look at some problems that can occur and some links to additional information.

The examples in this article use a test certificate, not a production certificate from a Certificate Authority. For purposes of this article, I thought a test certificate – also referred to as a *self-signed certificate* – was best since not everyone has a certificate. It also teaches you how to generate a certificate for testing. Regardless, the procedures described here apply exactly the same when you're using a real certificate. When you have a real one, you can just skip the parts about generating a test certificate.

The software used in this article was:

- IE4.0 40bit
- NS Communicator 4.51 with SmartUpdate
- MS Personal Web Server (create a virtual directory called "sample" to run samples)
- MS Windows NT 4.0 SP4

In a typical scenario visitors to a Web page download any elements into their browser cache. This could include the page itself, and some images as well as applets. Depending on browser settings, the next time users visit the page all of these elements may be reloaded from the Web site. This is certainly feasible for elements such as HTML and images, but can be time-consuming and frustrating for users of your applets. Internet Explorer and Netscape have different approaches to the actual implementation of downloading and installing, but they also share some common ideas.

Digital Certificate and Signatures

The use of digital signatures and certificates to verify the authenticity of software is nothing new. Software downloaded over the Internet provides the user no assurance about the authenticity of the author, where the software came from or when it was created. A common metaphor for a digital certificate used in code signing is "shrink-wrap." Software bought off the shelf at your local retailer is packaged with labels and markings indicating the company of origin. Software downloaded over the Internet has no such visible labels or markings.

DIRECT IE AND NS TO INSTALL YOUR APPLETS AND CLASSES PERMANENTLY

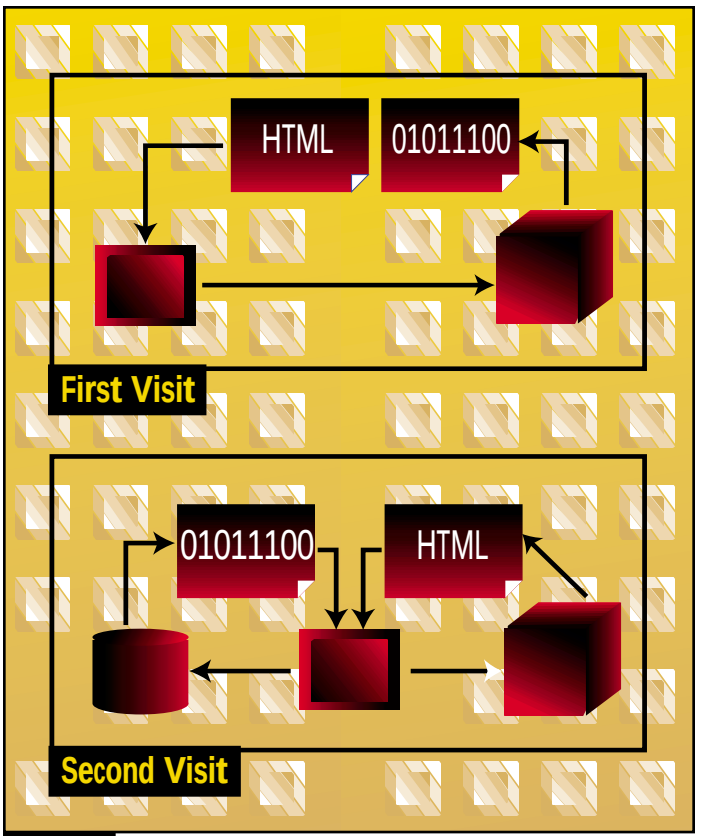


FIGURE 1 Classes that are premanently installed loaded locally after first visit

That's where digital certificates and signatures come in. The combination of digital certificates and signatures provides the labeling or shrink-wrapping that supplies information such as author name, time created and expiration date; they also ensure that the software wasn't tampered with since it was signed. A third party – a Certificate Authority (CA) – normally issues certificates. Basically, the Certificate Authority vouches for the identity of the individual using the certificate to sign his or her software. Popular CA's are Verisign and Thawte. You may also be able to obtain a certificate from your own company, if they're set up to issue certificates.

Netscape and Internet Explorer certificates are based on the standard x509, but deviate after that to produce certificates that are incompatible with each other. Internet Explorer has its Authenticode certificates; Netscape, its Object-Signing certificates.

Packaging

The use of packaging mechanisms to bundle your code and supporting files is common in both browsers. Internet Explorer uses the CAB format and Netscape uses the JAR format. This provides for quicker download times and is a requirement if you want to use managed installation mechanisms provided by Internet Explorer and Netscape Communicator. The former recognizes the JAR format but is unable to recognize a digital signature within. The latter doesn't recognize the CAB format.

Tools

There are utilities for both browsers that allow developers to generate self-signed digital certificates as well as packages, and sign their code. The first five utilities described here are for Internet Explorer only. The last one is for Netscape Communicator only. See the Links section for tool locations. If you have the Microsoft SDK for Java, you may already have the Microsoft utilities.

- ***MAKECERT***: Generates a test x509 certificate
- ***CERT2SPC***: Generates a test Software Publishing Certificate from an x509 certificate
- ***SIGNCODE***: Signs code using a Software Publishing Certificate for IE
- ***CABARC***: Builds a cabinet file of specified files
- ***DUBUILD***: Builds a cabinet and OSD and is an all-in-one tool. If you're uncomfortable with XML or don't wish to build your OSDs, this tool is for you. If you're up to it, you can also code the OSD by hand
- ***SIGNTOOL***: Packages and signs code using an object-signing certificate for Netscape; also used for generating test object-signing certificates

Scripted Install Procedure

The Microsoft virtual machine and its Java Package Manager use an XML-based description language, Open Software Description (OSD), that directs the Java Package Manager on how to handle different aspects of the download (see Listing 1). An OSD is also used to describe any dependencies between various components of the download. Netscape, on the other hand, uses a combination of JavaScript and Java in the package "netscape.softupdate.*" to allow developers to write scripts that control the flow of the installation (see Listing 2). Each of these procedures allows developers to control their installation by querying properties such as:

- *OS version*
- *Browser version*
- *Software version*
- *Ensure proper space exists before install*
- *Dependencies between different Java packages and versions*

Depending on which procedure you're using, some of these properties may not be available.

Managed Installation

Both browsers have an installation manager available to perform the actual install. Internet Explorer uses the Java Package Manager. Netscape uses the JAR Installation Manager and a feature called SmartUpdate. These installation managers allow the developer to install and update software automatically.

JAVA PACKAGE MANAGER

The Java Package Manager was introduced in Internet Explorer 4.x and provides the following features:

- ***Version Control***: Enables you to update older versions of your software and ensure that software isn't downgraded. *Note*: You can't downgrade your software with the Java Package Manager.
- ***Namespaces***: Prevents collisions between same-named packages. Before namespace, packages were installed into the CLASSPATH and provided no protection for libraries that may have had identical package names. By providing a namespace for each installed package, you avoid this collision. There's also a global namespace. Packages installed into the global namespace are accessible to all Java Packages. This would be ideal for a generic library that other installed applications could use.
- ***Improved Security***: Fine-grained security is now possible. The Java Package Manager requires that packages be signed with the Java Permissions to step out of the sandbox. With these permissions you can control access to the UI, the file system and other system resources such as sockets and threads. You can use the default permissions provided by the different levels (high, medium and low) or you can specify a custom permissions file.

  You can view already installed packages by opening the "Downloaded Program Files" folder in your Windows Explorer. You can also see this same list using Internet Explorer. Open View—>Internet Options—>Settings—>View Objects. Unlike Netscape, once a package is installed, you can begin using it immediately.

JAR INSTALLATION MANAGER/SMARTUPDATE

The JAR Installation Manager and SmartUpdate were introduced in Netscape 4.x and provide features similar to those in the Java Package Manager. Some features of SmartUpdate may not be available, depending on the version of Communicator used.

- ***Version Control***: As with the Java Package Manager, you can update older versions of your software and ensure that it isn't downgraded. However, SmartUpdate also gives you a couple of advantages over the features provided by the Java Package Manager: (1) you can downgrade previously installed packages, and (2) you can force an install despite the version. SmartUpdate maintains this in the Client Version Registry. Unlike the Java Package Manager, when you install a package with SmartUpdate, you must restart Communicator before you can use it. Your install script should indicate this with a dialog.
- ***Improved Security***: Fine-grained security is now possible. The JAR Installation Manager requires packages to be signed in order to be installed and step out of the sandbox. One noticeable difference between IE and NS is that IE's permissions are encoded with the digital signatures, whereas NS requires the use of the "Capabilities" API to request permissions at runtime.
- ***Registry of Installed Applications***: Netscape provides a Client Version Registry area that records all installed software registered for use with Communicator.

Sample Application

Now that we have some of the basics behind us, let's start applying it. We're going to install a basic clock applet (see Listing 3), then write an HTML page that demonstrates the installed applet. Please note the package statement in the source. The Java Package Manager requires you to place your classes into a package. Once you've compiled the source, we'll start with the process for Internet Explorer, then Netscape Communicator.

Internet Explorer

1. ***Generate a test x509 certificate.***

   The options used for this step are:

- ***-sk:*** Key Name
- ***-n:*** Certificate Subject x500 Name (i.e. CN=My Name)

```
makecert -sk SampleKey -n "CN=TestCertificate" SampleTestCert.cer
```

2. ***Turn an x509 certificate into a Software Publishing Certificate.***

```
cert2spc SampleCert.cer SampleTestCert.spc
```

3. ***Create your distribution unit.***

   The options used for this step are:

- ***/D:*** Distribution unit name or "friendly name"
- ***/I:*** Include files matching this pattern
- ***/V:*** Version number for distribution unit

```
dubuild sample.cab . /D "Sample Application" /I *.class /V 1,0,0,0
```

By not using the /N option, we're placing our package into the global namespace. To get an idea what the OSD generated by dubuild looks like, open your CAB and view the generated OSD file. It should look like the one in Listing 1.

4. ***Sign your code.***

During this step you'd normally supply an additional parameter, –t, to indicate a URL to a time-stamping CGI on your Certificate Authority's Web site. However, due to firewall considerations, this may be impossible. For test purposes this isn't a problem. You'll see a message indicating the CAB has been signed but not time-stamped.

The options used for this step are:

- ***-j:*** Indicates the source of the Java permissions
- ***-jp:*** A parameter to pass to javasign.dll. In this case, the security level of medium
- ***-spc:*** The software publishing certificate generated in step 2
- ***-k:*** The key generated from step 1

```
signcode -j javasign.dll -jp medium -spc SampleTestCert.spc -k Sam-
pleKey sample.cab
```

With our code packaged and signed, we're ready to deploy. We'll use the <APPLET> tag with three parameters:

- ***useslibrary:*** Specifies a name for the distribution unit, which should match the one you specified when you ran dubuild
- ***useslibrarycodebase:*** Specifies the codebase for the distribution unit CAB file
- ***useslibraryversion:*** Specifies the version number, which should match the one you specified when you ran dubuild

```
<APPLET CODE=com.mysample.application.Sample HEIGHT=20 WIDTH=100>
<PARAM NAME=useslibrary VALUE="Sample Application">
<PARAM NAME=useslibrarycodebase VALUE="sample.cab">
<PARAM NAME=useslibraryversion VALUE="1,0,0,0">
</APPLET>
```

These parameters will be ignored in Internet Explorer version 3 and Netscape browsers. Once you've saved the page, open it in your browser. If everything went okay, you should see a Security Warning dialog. Click on "Yes" to trust; you should then see the sample applet start.

VERIFY INSTALLATION

Open the "Downloaded Program Files" folder in Windows Explorer or do a View—>Internet Options—>Settings—>View Objects and you should see the same display (see Figure 2). Behind the scenes the Java Package Manager is also updating the registry entries for your distribution unit. Run regedit and open "HKEY_LOCAL_MACHINE\Software\Microsoft\Code Store Database\Global Namespace". You should see keys for all packages installed in the global namespace, top-level first. Open the "com" key and you'll see the "mysample" key underneath it.

Netscape Communicator

We'll now examine the process for Netscape Communicator. *Note*: Please shut down Communicator before running step 1 or you risk corrupting Communicator's security database. You'll also note that the signing and packaging steps are combined. To create and store an object-signing certificate, you'll also need to have a password set. If you don't, step 1 will fail. To set a password, open the Security Window and select "Passwords." As you'll see shortly, to permanently install Java Packages to run locally, you must place your signed JAR of classes inside another JAR containing an install script (see Figure 3). Wherever you specify the path to the certificate directory, you'll need to replace the path of the certificate database to match the path on your system.

1. ***Create test object-signing certificate.***
   The options used for this step are:

- ***G:*** The nickname of our object-signing certificate
- ***-d:*** The location of the certificate database

```
signtool -G"SampleNetscapeObjectCert"
-d"e:\progra~1\netscape\users\default"
```

When you run signtool you'll be prompted for a number of parameters, as it states in the message before it runs. These are optional except for the database password. You can bypass them by pressing "Enter." To verify your certificate has been added, type the following command:

```
signtool -L -d"e:\progra~1\netscape\users\default"
```

You should see a list of Certificate Authorities, including yours, with asterisks beside them. The asterisk means that this certificate can be used to sign objects. If you decide not to install directly into Communicator, you can import your certificate by placing a link to it on a Web page, then clicking on it. You'll be guided through a series of dialogs to install it. If you're distributing this test object-signing certificate from a Web site, you'll also need to configure a MIME entry for it on the Web server you're using. You can also start Communicator and look at the Security Info Window. Click on "Signers" under "Certificates." You may have to scroll until you see "SampleNetscapeObjectCert".

2. ***Create install script (see Listing 2)***.
   The install script's purpose is to direct the actual install.

3. ***Create and sign Software JAR***.
   The options used for this step are:

- ***b:*** Specifies the base filename for the .rsa and .sf files
- ***d:*** Specifies the location of the certificate directory
- ***k:*** Specifies the nickname of the test object-signing certificate created in step 1
- ***Z:*** Directs signtool to create a JAR with the specified name

```
signtool -b "Sample Application"
-d"e:\progra~1\netscape\users\default" -k SampleNetscapeObjectCert
-Z sample.jar .
```

4. ***Create and sign Install JAR***.
   Now we'll create the install JAR that will hold our install script and software JAR. You'll be prompted for the certificate database password. You can also use the "-p" option to specify the password. I'd recommend using this option only during testing as the password is visible.

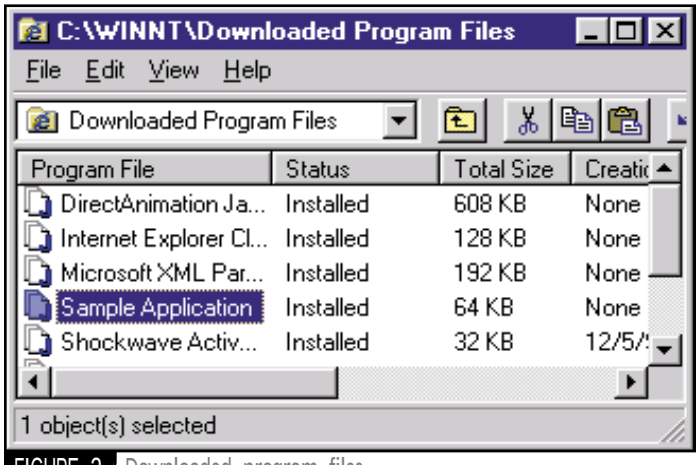


FIGURE 2 Downloaded program files

FIGURE 3 Netscape requires your software to be packaged inside another JAR with install script.

- ***b:*** Specifies the base filename for the .rsa and .sf files
- ***i:*** Specifies the name of the install script
- ***d:*** Specifies the location of the certificate directory
- ***k:*** Specifies the nickname of the test object-signing certificate created in step 1
- ***Z:*** Directs signtool to create a JAR with the specified name

```
signtool -b sampleinst -i sample.js
-d"e:\progra~1\netscape\users\default" -k SampleNetscapeObjectCert
-Z sampleinst.jar .
```

Now that the classes are packaged and signed, we can deploy. We'll be using what's referred to as a *trigger script* (see Listing 4). Once you've saved the page, open it in your browser. If everything went okay, you should see a Security Warning dialog. Click on "Grant" to trust, then you'll see a dialog indicating Communicator must be restarted before using the new classes.

Verifying Installation

Start Communicator and open Edit—>Preferences—>Advanced—> SmartUpdate. You'll see your package name in the listing of installed software (see Figure 4).

Using the Installed Software

Let's try out the newly installed package. Put the following code into a page and bring it up in your browser (see Figure 5). Notice there is no CAB base or archive parameter specifying the cabinet or JAR where the classes are to be found. That's because the classes are being loaded locally by the browser.

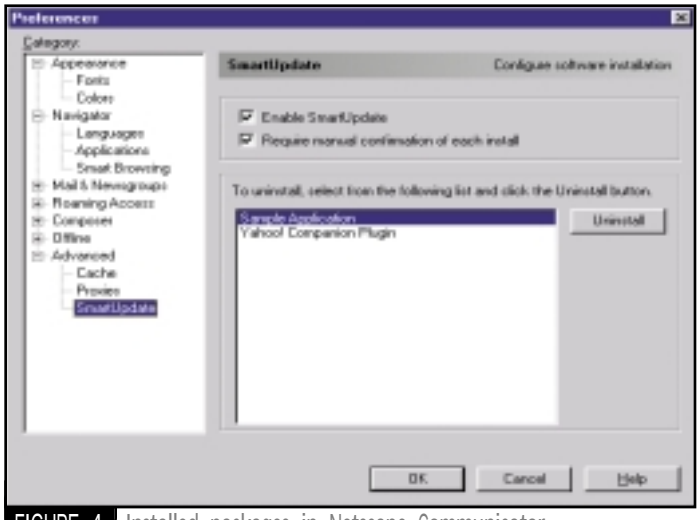


FIGURE 4 Installed packages in Netscape Communicator

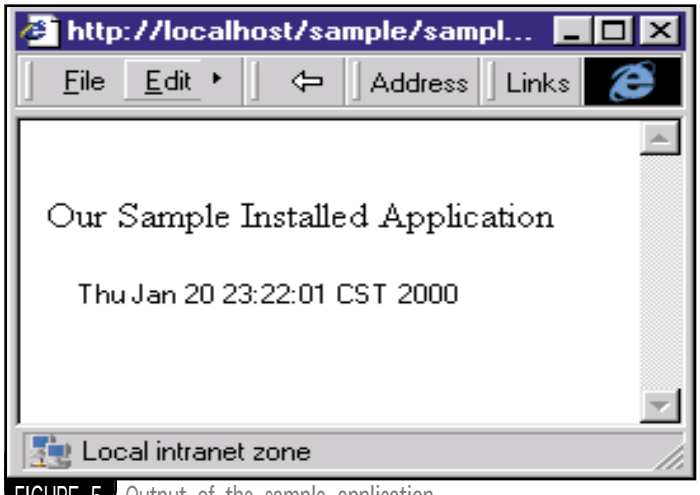


FIGURE 5 Output of the sample application

```
<HTML>
<BODY>
<H1> Our Sample Installed Application </H1>
<APPLET CODE=com.mysample.application.Sample HEIGHT=20
WIDTH=200></APPLET>
</BODY>
</HTML>
```

Updating Your Software

Eventually most software needs to be updated. New hardware, bugs or new versions can create this need. With the Java Package Manager and Netscape with SmartUpdate, you can update the version a user has installed with the same ease. Simply increment the version numbers appropriately and update your HTML page and/or the install script. The next time users visit the page, they'll be prompted to install a new version if the version you specified is newer than the installed version.

Problems

Sometimes things go awry. You may see messages indicating security failures or you may see nothing on the page. There are tools to aid in diagnosing download failures. CODLLGVW is a utility that examines the code download error log created during download. Netscape includes a host of error codes that could arise during SmartUpdate. In Internet Explorer one of the most common is that there may be insufficient disk space to install your component. Another possible area is errors in the OSD. The CDF utility can be used to find possible errors in your OSD, such as missing or misspelled tags. Another problem I've seen is not specifying a package in your OSD that's in your CAB. Also, make sure the name you use in the useslibrary tag matches the friendly name of your distribution unit. And when using IE, make sure the version numbers match on your useslibraryversion tag and the one you specified when you ran DUBUILD.

Summary

As you can see, distributing your code for permanent installation isn't difficult. And I'm sure the users of your applet will appreciate the decreased loadtime. Although the process for each browser is somewhat different, it's similar overall.

The following links should provide any further details you may need. Good luck!

1. Packaging Components for Internet Distribution: http://msdn.microsoft.com/workshop/delivery/download/tutorials/buttons_download.asp
2. Deploying Java in Internet Explorer and Netscape Communicator: http://support.microsoft.com/support/kb/articles/q179/6/52.asp
3. Downloading Code on the Web: http://msdn.microsoft.com/workshop/components/downcode.asp
4. Object Signing – Establishing Trust for Downloaded Software: http://developer.netscape.com:80/docs/manual/signedobj/trust/owp.htm
5. Object Signing Resources: http://developer.netscape.com:80/docs/manuals/signedobj/overview.html
6. SmartUpdate Developers Guide: http://developer.netscape.com:80/docs/manuals/communicator/jarman

Author Bio

Mike Jasnowski, a Sun-certified Java programmer, has 17+ years of programming experience and 3+ years with Java. He works for a software company in Kansas City, Missouri.

mjasnowski@cerner.com

Idea Integration

www.idea.com

Listing 1

```
<?XML version="1.0"?>
<!DOCTYPE SOFTPKG SYSTEM "http://www.microsoft.com/stan-
dards/osd/osd.dtd">
<?XML::namespace
href="http://www.microsoft.com/standards/osd/msicd.dtd"
as="MSICD"?>

<SOFTPKG NAME="Sample Application" VERSION="1,0,0,0">
<!-- created by DUBuild version 5.00.3023 -->

    <TITLE>Sample Application</TITLE>

    <MSICD::JAVA>

        <PACKAGE NAME="com.mysample.application" VER-
         SION="1,0,0,0">
            <IMPLEMENTATION/>
        </PACKAGE>

    </MSICD::JAVA>

</SOFTPKG>
```

Listing 2

```
//* Sample JavaScript installation script

// Make sure Java is enabled before doing anything else

if (navigator.javaEnabled()){

    answer = confirm("Do you want to install Sample Application?");

    if (answer){

      // Create a version object and a software update object

      vi = new netscape.softupdate.VersionInfo(1,0,0,0);
      su = new netscape.softupdate.SoftwareUpdate(this,"Sample
Application");

      // Start the install process
      err = su.StartInstall("java/download/Sample
Application",vi,netscape.softupdate.SoftwareUpdate.LIMITED_INS
TALL);

      if (err == 0){
   // Find the Java Download directory on users computer
   JavaFolder = su.GetFolder("Java Download");

   // Install the JAR File. Unpack it and list where it goes
   err = su.AddSubcomponent("Sample
Application",vi,"sample.jar",JavaFolder,"",this.force);
      }

      if (err != 0){
    alert(err);
      su.AbortInstall();
    alert("Installation Aborted.");
      }else{
    su.FinalizeInstall();
    alert("Installation complete. You must restart Communica-
    tor to use the Sample Application");
      }
    }
}
```

Listing 3

```
package com.mysample.application;

import java.applet.Applet;
import java.awt.Graphics;
import java.lang.Runnable;
import java.util.Date;
import java.awt.Color;

/**
*
* Sample.java
*
* @author Mike Jasnowski
* @version 1.0
*/
public class Sample extends Applet implements Runnable{

      private String time_string;
      private Thread runner = null;

      public void init(){
             runner = new Thread(this);
             runner.start();
      }

      public void run(){
          while (true){
                time_string = new Date().toString();
                repaint();
                try {
                         runner.sleep(100);
                } catch(InterruptedException e){}
          }
      }

      public void paint(Graphics g){
          setBackground(Color.white);
          g.drawString(time_string,12,12);
      }

}
```

Listing 4

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">

// Ensure Java is enabled

if ( navigator.javaEnabled() ) {

   // Create a Trigger Object

   var trigger = netscape.softupdate.Trigger;

   // Create a VersionInfo object

   version = new netscape.softupdate.VersionInfo(1,0,0,0);

   // Make sure SmartUpdate is available

   if ( trigger.UpdateEnabled() ){

 // Call ConditionalSoftwareUpdate pointing to Install JAR

        trigger.ConditionalSoftwareUpdate ("http://local
        host/sample/sampleinst.jar","java/download/Sample
        Application",version,trigger.DEFAULT_MODE);
   }else
          alert("Enable SmartUpdate before running this
           script.");
}
    else
       alert("Enable Java before running this script.");

</SCRIPT>
</HEAD>
<BODY>
</BODY>
</HTML>
```

Beware the Daemons

Avoid damnation with careful use of daemon threads

WRITTEN BY
TONY LAPASO

Have you ever wondered why it's necessary to call System.exit() to force your Java UI application to exit? Have a look at the code in Listing 1.

After the setVisible() method has been called at line 15, the main() method completes execution. Why doesn't the program end at that point?

The answer to this question is rooted in a subtlety of Java's thread creation mechanism.

Java makes a distinction between a *user* thread and another type of thread known as a *daemon* thread. The difference between these two types of threads is straightforward. If the Java runtime determines that the only threads running in an application are daemon threads (i.e., there are no user threads), the Java runtime promptly closes down the application. On the other hand, if at least one user thread is alive, the Java runtime won't terminate your application. In all other respect(s) the Java runtime treats daemon threads and user threads in exactly the same manner.

When your main() method initially receives control from the Java runtime, it executes in the context of a user thread. As long as the main-method thread or any other user thread remains alive, your application will continue to execute.

To see just how a thread becomes a daemon thread, read on.

AUTHOR BIO

Tony LaPaso is a Java application specialist and a Sun-certified Java programmer working for Greenbrier & Russel, Inc. (www.gr.com), in Phoenix, Arizona. His particular area of interest is distributed application development using RMI, EJBs, servlets and JSPs. Tony holds a master's degree in computer science from Arizona State University.

Going to the Devil

There are two ways a thread can become a daemon thread (or a user thread, for that matter) without putting your soul at risk. First, you can explicitly specify a thread to be a daemon thread by calling setDaemon(true) on a Thread object. Note that the setDaemon() method must be called before the thread's start() method is invoked, as the following snippet shows:

```
1: Thread t = new Thread() {
2:     public void run() {
3:         System.out.println("I'm a
           man of wealth and
           taste...");
4:     }
5: };
6:
7: t.setDaemon(true);
8: t.start();
```

The second technique for creating a daemon thread is based on an often overlooked feature of Java's threading behavior. If a thread creates a new thread and doesn't call setDaemon() on the new thread before it's started, the new thread will inherit the "daemon-status" of the creating thread. For example, unless setDaemon(false) is called, all threads created by daemon threads will also be daemon threads; likewise, unless setDaemon(true) is called, all threads created by user threads will be user threads.

The program in Listing 2 illustrates this behavior. The program creates a thread, t1, whose only purpose in life is to create another thread, t2, whose destiny is to print a greeting, "Pleased to meet you," for all eternity. If you run this code you'll see that even though the main thread ends, t2 goes on forever, printing its tempting message. This behavior is due to the fact that t2 inherited its user thread status from its creator, t1. t1 in turn inherited *its* user thread status from its creator, the main-method thread, which, as I said, is always created by the Java runtime as a user thread.

By uncommenting line 18, t1 becomes designated as a daemon. When t1 creates t2, it too becomes a daemon thread. As a result, when the main-method thread exits, the Java runtime will end the application, since the only remaining thread (t2) is a daemon. This may or may not be the functionality you wanted or expected.

Exorcise the Daemons

Applying what we've learned about daemon threads, let's consider again why it's usually necessary for a Java graphical application to call System.exit() in order to end. As it turns out, the *first time* an application makes a user-interface component (e.g., Frame) visible, Java's internal AWT plumbing creates several threads behind the scenes. These threads are responsible for handling event queue management and window repainting. Typically, the first UI component made visible is a Frame or Dialog (or one of their JFC derivatives) via a call to setVisible(true). If the call to setVisible() is made from within a user thread (such as the main-method thread), these new AWT threads will *also* be user threads and won't be shut down simply because the main-method thread ends. This explains why your user interface is visible even after the main-method thread ends.

Let's put what we've learned to the test. The program in Listing 3 creates a Frame from within a daemon thread. After the daemon thread is created, the main thread goes to sleep for five seconds and then ends. Once the main-method thread ends, there are no longer any user threads, so the Java runtime shuts down the application.

Don't Sell Your Soul

The moral of this story is that daemon threads should be used judiciously. They were designed to be used as servants to their user thread masters. When no more user threads exist, daemon threads lose their reason for living and the Java runtime steps in and mercifully ends their benign existence.

Because the life of a daemon thread can be a precarious one, you should be careful with the sort of tasks you assign to them. A somewhat contrived yet illustrative example of their perils is a daemon thread dedicated to opening a log file, appending to it, then closing the log file at a predetermined interval. When the Java runtime determines that all user threads have ended, it will quickly kill the daemon logging thread, possibly resulting in an inconsistent log file. To avoid this it's best to structure your code so that any work assigned to a daemon thread will be completed before all the user threads die. So don't give in to temptation – be careful using daemon threads...and behave yourself.

tony@absoluteJava.com

Listing 1

```
 1: import java.awt.*;
 2: import java.awt.event.*;
 3:
 4: public class Tester {
 5:     public static void
        main(String[] args) {
 6:         Frame f = new Frame();
 7:
 8:         f.addWindowListener(new
            WindowAdapter() {
 9:             public void windowClos-
                ing(WindowEvent evt) {
10:                 System.exit(0);
11:             }
12:         });
13:
14:         f.setBounds(30, 30, 400,
            400);
15:         f.setVisible(true);
16:     }
17: }
```

Listing 2

```
 1: public class Tester {
 2:     public static void
        main(String[] args) {
 3:         Thread t1 = new Thread() {
 4:             public void run() {
 5:                 // Fire off a new
                thread.
 6:                 Thread t2 = new
                    Thread() {
 7:                     public void run()
                        {
 8:                         while(true)
 9: System.out.println("Pleased to
    meet you.");
10:                     }
11:                 }; // end of t2
12:
13:                 t2.start();
14:             }
15:         }; // end of t1
17:
18:         // t1.setDaemon(true);
19:         t1.start();
20:
21:         for(int i = 0; i < 50;
            ++i)
22: System.out.println("Hope you
    guess my name.");
23:
24:         System.out.println("Main
            thread is about to end.");
25:     } // main()
26: } // Tester
```

Listing 3

```
 1: import java.awt.*;
 2:
 3: public class Tester {
 4:     public static void
        main(String[] args) {
 5:         Thread t1 = new Thread() {
 6:             public void run() {
 7:                 Frame f = new
                    Frame();
 8:                 f.setBounds(30, 30,
                    400, 400);
 9:                 f.setVisible(true);
10:             }
11:         }; // end of t1
12:
13:         t1.setDaemon(true);
14:         t1.start();
15:
16:         try { Thread.sleep(5000);
            }
17:         catch(InterruptedException
            e) {}
18:     } // main()
19: } // Tester
```

Interview...

with CHARLES STACK PRESIDENT AND CEO OF FLASHLINE.COM

JDJ: What has Flashline been up to since JavaOne?
Stack: We've been enhancing our marketplace and still offer several hundred software components for resale, primarily JavaBeans. Now we also have some Enterprise JavaBeans available. We expect to add a significant number of those in the next three months. People can come to our site, look over the available components, pick the ones they want and get them downloaded directly to their PC.

JDJ: You were one of the good reference sites for JavaServer pages.
Stack: Yes, we're like a poster child for JavaServer Pages.

JDJ: And how's that going?
Stack: Very well. We built the entire site using JavaServer Pages. It's a very sophisticated e-commerce site with real-time credit card authorizations and the multiple back ends. People can enter coupons and get discounts. The vendors can post their own products, look at real-time sales information, get instant registrations from purchases, instant registrations from downloads and look at sales charts. They can do bundling on their own. The JavaServer Pages' architecture has been very flexible for us.

JDJ: Now you're the man who will really tell us what's selling.
Stack: We've seen significant growth quarter to quarter, in excess of 50% a quarter, so that's a good sign. I think the big market is server-side components, which is still developing. As I said, we have some Enterprise JavaBeans now. We'll probably quintuple that number in the next 60 days. I think the server-side market, if you look at the user interface market, is probably just a fraction of what the server-side market is going to be, because frankly, how many drop-down boxes do you need?

JDJ: Can a normal developer come along and submit a bean for you to distribute?
Stack: We have two excellent programs specifically targeted for developers that aren't quite developing commercial quality, off-the-shelf components. We have a components-by-design service. We have about 1,200 developers registered for that, and people who are looking for specific components can come in and post requests, and at that point it kind of functions like a cross between natch.com and eBay dating for developers, except you get an auction model thrown in there. People post requests including UML diagrams, raw interface specifications and extremely detailed RFPs. And then an e-mail goes out to the developers in their areas and they can come in. Then there's an interactive question-and-answer time frame where people can post questions and further refine the actual deliverable. At some point, after the questions and answers close, the products or the requests are tightly defined, the developers bid on that project and the requester looks at their qualifications, the price, the work they've done, the feedback that has come in on their performance in the past, and selects a winning developer. It's an excellent program for somebody who doesn't have commercial components but still wants to do Java development.

JDJ: And who pays for that?
Stack: We currently charge a nominal fee for the request. It's $100 per request. Other than that, it's a completely free service.

JDJ: The developer doesn't pay anything?
Stack: There's no fee to the developer at all.

JDJ: That's really nothing for a finder's fee, isn't it?
Stack: The $100 is there to make sure that people take the quality of the requests and the quality of what people are looking for seriously. We also have a program called BetaBeans, a free third-party service where developers who don't quite have commercial quality components can post them on our site and get them exposed to the thousands of daily visitors who can then download them for free and test them out. We have a built-in feedback mechanism where they can post requests for enhancements or bug reports that the developer can respond to and bring the quality of their components up to a commercial quality that could then go into the marketplace. Both of those programs are really designed to create more components.

JDJ: How do you, from Flashline's point of view, make sure that a component is up to a commercial standard?
Stack: We're in the process right now of developing certification standards, and we're going to proceed on two fronts. The first one is cross-application server compatibility testing, because in surveying our customers, the number one thing they wanted to know was that if they bought a component, it would work on this application server, and that application server wouldn't have to be wedded or bound to a particular product. The second aspect of the certification is more along the lines of what you're asking about. It's a certification program where we'll go into the component and validate that it meets certain minimum documentation standards. Part of that documentation will be performance testing metrics and UML diagrams to document that the component is of commercial quality. What we're not going to do in the near term is actually say this is good or this is bad. That's a pretty subjective area that, at least for now, we're going to stay out of. But we'll guarantee that it has a certain minimum level of documentation and will run on *X* number of applications.

"I think the big market is server-side components, which is still developing."

JDJ: Can we go to www.flashline.com for more information about everything you've just said today?
Stack: Yes. There's one more thing we announced today. It's an XML version of Javadoc called JavaDox. If you've ever used Javadoc, you know it spins out a large number of little files in an HTML format. We've modified the Javadoc programmer, actually enhanced it, with a doclet that spins out a single XML file for the Javadoc communication. This vastly improves the utility of Javadoc because you have a single file that's much more portable because it's a single file. It's searchable because it's in XML, and you can use stylesheets to reform it into any particular documentation standard you need for any purpose. And that's free. It's a Java doclet that's available at our site. Actually, it's at www.component-registry.com and there's a JavaDox program there.

VisualCafé Enterprise Edition for WebLogic

Making EJB development easy!

WRITTEN BY
JASON WESTRA

When I started working with Java, I mentioned my move to a colleague of mine, a Microsoft devotee. He wasn't willing to move to the Java platform until supporting integrated development environments (IDEs) were as powerful and easy to use as Visual Basic. Although at the time nothing in the Java world was as simple or configurable as Visual Basic, I bit the Java bullet – and the bullet tasted like VisualCafé. Originally from Symantec Corp. (www.symantec.com) but now owned by an independent company created by Warburg, Pincus and BEA Systems, VisualCafé was the closest Java IDE in the industry that could compare to VB, and it remains on the bleeding edge of support for new Java technologies. This month in EJB Home I'll discuss what to look for in an IDE that supports EJB, as well as the support for Enterprise JavaBeans development that has been integrated into the VisualCafé Enterprise Edition for WebLogic.

What Does an EJB IDE Look Like?

An IDE is just that – an integrated development environment. It's a seamless, interoperable toolset that takes you through development, debugging and deployment without leaving the familiar confines of your development product of choice. When looking for an IDE for EJB development there are four items to keep in mind:

1. *An open API to encourage integration with other EJB tools*
2. *Support for rapidly developing components*
3. *Easy deployment of these components*
4. *Robust distributed debugging facilities to test your components*

Essentially, you should never have to leave your development environment to perform the necessary steps of coding, deploying and debugging an enterprise bean.

Other things to look for in an IDE include integrated source code control for team development and support for multiple JDKs for testing your components (which won't be covered here).

Why Use VisualCafé Enterprise Edition for WebLogic?

A problem that continues to hinder Java development is the lag between the introduction of a new technology and the IDE support for rapid application development with the technology. Examples of this lag have been seen in product support for each JDK release as well as Swing, JavaBeans and, most recently, Enterprise JavaBeans (EJB). As previously mentioned, VisualCafé has always been a front-runner in the race to support new Java technologies, and its first-generation effort to integrate support for EJB technology is no different. Currently, VisualCafé provides an integrated development environment for building EJB applications with the WebLogic Server. While the product could be expanded to support other application servers, I doubt this will come to fruition. The new owners of VisualCafé will most likely orient this tool toward an even tighter integration with the WebLogic Server.

The following sections show how the VisualCafé Enterprise Edition for WebLogic supports the key features of an EJB IDE, and why this product combination may be a good fit for your next EJB effort.

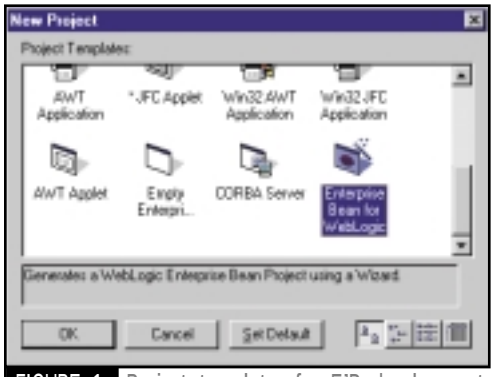


FIGURE 1 Project templates for EJB development

OPEN API

VisualCafé has an open API to allow vendors to integrate their products and tools into its environment. The open API encourages vendors to build tools that seamlessly interact with VisualCafé to offer you a more positive experience. Today numerous EJB products have been integrated into VisualCafé, including InLine Software Corporation's InLine Standard product (www.inlinesoftware.com), various UML modeling/code generation plug-ins and third-party-distributed debugging software. For instance, Inline's UML Bridge product performs real-time code generation and reverse engineering between Rational Rose models and VisualCafé's code editor.

While many application servers have proprietary development environments integrated closely with their server, VisualCafé and WebLogic have stood by their best-of-breed approach toward tool integration.

EJB CODE GENERATION

VisualCafé for WebLogic offers two project templates for EJB component development (see Figure 1). One is an empty EJB project into which you can import existing EJB components, while the second is for developing new EJB components in rapid fashion utilizing code generation.

The enterprise bean for the WebLogic project template actually consists of two projects: a server project for the enterprise bean component and a client project containing a generated test client for the bean. The template prompts you to use the enterprise bean for WebLogic Wizard to enter information about the enterprise bean you wish to create. Afterwards, the wizard generates the appropriate EJB classes and VisualCafé XML descriptor for your component. (*Note*: The XML descriptor is not the EJB 1.1 format, but a proprietary format for internal use only.) If you choose to create a session bean, the remote and home interfaces would be generated as well as a stubbed skeleton of the enterprise bean. Choosing to create an entity bean will generate these classes, plus the primary key class for the bean.

Code generation of your beans reduces errors in coding your component classes to the EJB specification and speeds development through the automation of mundane coding. Likewise, a generated test client facilitates the testing process of your EJB component, allowing you to focus your efforts on developing business components – not the harnesses that test them!

DEPLOYMENT OF EJB COMPONENTS

VisualCafé for WebLogic allows you to configure an enterprise bean, deploy it to a WebLogic Server and start the server without leaving the VisualCafé environment. It even supports hot deploy capabilities into a running WebLogic server.

The auto-deploy process is as follows: upon selecting Configure Bean Descriptor (see Figure 2), VisualCafé configures an XML descriptor to hold deployment descriptor information in a portable fashion. This XML descriptor is used by VisualCafé's configuration tool to create a serialized deployment descriptor for WebLogic and to generate the EJB container classes needed by WebLogic to support the enterprise bean.

With the container classes and serialized descriptor generated, you can deploy the bean to the server from within VisualCafé. Choosing Deliver Enterprise Bean to WebLogic Server from the Project menu will JAR the necessary files and add the JAR file to the weblogic.properties file, thereby deploying the bean to the server.

Start WebLogic Server and Shutdown WebLogic Server manage the WebLogic Server without the need for the WebLogic Console or command-line start-up scripts, or even the need to double-click a shortcut to launch it. The deployment process never forces you to leave the VisualCafé' environment.

DISTRIBUTED DEBUGGING

Most early adopters of distributed Java and EJB can tell endless fireside stories about the thousands of printlines they had to code in order to solve the simplest of coding errors. Now, with VisualCafé, you can say good-bye to printlines! VisualCafé has advanced support for debugging distributed objects, including enterprise beans in the WebLogic Server.

Distributed debugging increases your efficiency by allowing you to step through code in a running EJB container as if the code executed locally on the same virtual machine as the VisualCafé's client program. VisualCafé displays all debugging information from a single console for easy tracking of application statistics during execution.

• • •

As you've seen, the VisualCafé Enterprise Edition for WebLogic has four qualities important in an IDE for developing EJB applications. Despite its new product features, however, it's not without faults, so let's take a look at some areas where it can be improved.

Improving VisualCafé for WebLogic

The first thing I noticed while using the Enterprise Edition for WebLogic was that the stability of VisualCafé is still questionable. Ever since I started working with the product, it has had problems with crashing at the most inopportune times. I call a certain type of VisualCafé crash a *ghost shutdown*. This is when VisualCafé eats a ton of CPU on your machine, then disappears without a trace! Because VisualCafé allows you to plug in different virtual machines, I always ensure that my compiler, client VM and server VM all match the JDK version. Read the Readme file for information on your development environment before wasting time trying to resolve a known incompatibility.

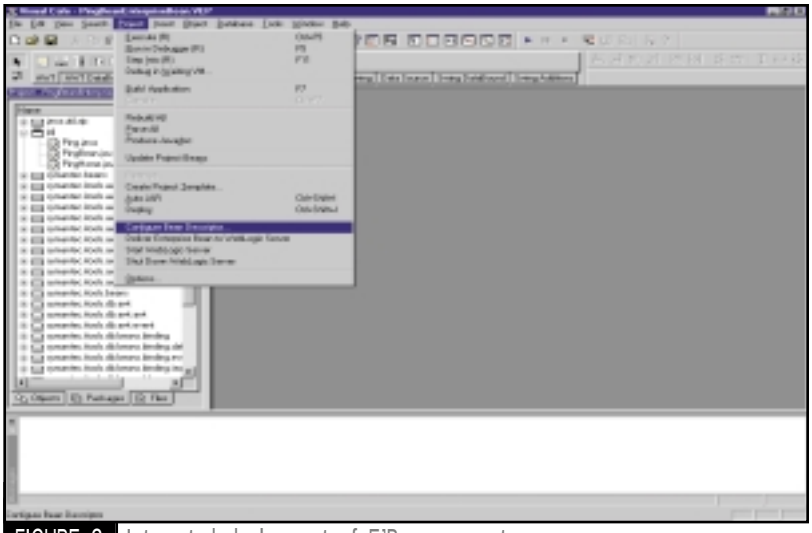


FIGURE 2 Integrated deployment of EJB components

Second, VisualCafé only allows EJB component generation through the enterprise bean for WebLogic Wizard after creating a new project template. There's no way to go to the wizard from an existing project, and a new server project per bean results in an explosion of VisualCafé projects to manage (remember – it actually creates two per bean: a server and a client project). I believe VisualCafé forces you to create a server project per bean because it made it easier for them to auto-deploy the bean's JAR file to the server. When the product becomes EJB 1.1 compliant, it'll have to support deploying multiple beans in a single JAR, but in the near term this is an inconvenience.

Third, when creating a new enterprise bean with the wizard, if you decide to change the name of any bean classes, you'll have major headaches finding all of the references to the old name that were generated by VisualCafé. For instance, changing the home interface name will force you to update the bean's XML descriptor manually. For those of you accustomed to VisualCafé's automatic change notification, this will be a sore spot, especially if you're new to EJB and not sure where to make the changes yourself.

Fourth, in release 3.1 a number of distributed debugging limitations still exist, especially around the Java 2 pluggable VM. For instance, some debugging features are disabled because VisualCafé doesn't support the functionality. Also, you can't define an application project that automatically executes remotely for VisualCafé's pluggable Java 2 VM. To do so you must manually transport files to the remote machine, start the remote process, attach to it and debug. This is a problem that needs to be addressed since many EJB developers are looking to use the JDK 1.2 and J2EE APIs, and conform to the latest EJB specification (1.1).

Last, VisualCafé Enterprise Edition for WebLogic is a memory hog! You'll experience slow performance from running VisualCafé and WebLogic Server simultaneously on your development machine. For development hardware I recommend at least a Pentium 450 MHz with 128MB RAM.

Conclusion

The widespread adoption of Enterprise JavaBeans as a server-side component model has led to the demand for advanced EJB development and deployment tools. VisualCafé Enterprise Edition for WebLogic met the demand with a powerful set of functionality focused on making EJB development easy. It has:

- An open API for customizing your environment
- Code generation wizards to speed development of your EJB components
- An auto-deploy feature that shelters you from the nuances of deploying components to the WebLogic Server
- Sophisticated, single-view debugging of distributed EJB components

Despite some growing pains, the VisualCafé Enterprise Edition for WebLogic is a good start for your team if WebLogic is your application server of choice.

AUTHOR BIO

Jason Westra is the CTO of Verge Technologies Group, Inc., a Java consulting firm specializing in e-business solutions with Enterprise JavaBeans.

westra@sys-con.com

Elixir

www.elixirtech.com

Picking the Right Development Tool

IMS uses JClass Components to build front end to testing system

WRITTEN BY
SAM WATTS

JClass LiveTable from KL Group proved to be the ideal solution for building an intuitive and dynamic graphical user interface for the Vanguard product family from Integrated Measurement Systems Inc., a global leader in the development of engineering testing systems.

Engineers involved in device testing know that intuition and on-the-fly interpretations have significant impact on device-testing sequences. The engineer's next step is often difficult to predict as it may depend on a real-time interpretation of the results produced by the immediately preceding test. As such, any system that's designed for use in these environments must be highly dynamic, allowing for seamless and interactive information flow between the device and the engineer. Flexibility, responsiveness and ease of use in the software interface help to ensure that fast, accurate and effective testing takes place.

IMS, based in Beaverton, Oregon, is well versed in the challenges posed by such a dynamic environment. IMS has developed Vanguard, an industry-leading family of hardware and software systems dedicated to the verification, characterization and failure analysis of complex, high-speed digital integrated circuits (ICs). By accelerating information exchange between the device and the engineer, Vanguard significantly increases engineering productivity and reduces the time required to get devices into production. For IMS customers these condensed testing phases translate into faster time to market and a powerful competitive advantage.

Underlying IMS's success in this field is a solid understanding of the way engineers think and work. Recognizing "the real-time analysis going on inside the engineer's mind," IMS focused on making Vanguard as intuitive, fast and responsive as possible. Vanguard's Java-based software architecture, together with the use of JClass Java components from KL Group, played a key role in realizing these goals. The graphical user interface had to be easy to read – and user-friendly – and allow engineers to set up and execute individual tests and test sequences quickly and easily.

Bob Vistica, senior software engineer at IMS and project lead on Vanguard's GUI development, spoke about some of the challenges he faced in developing the interface. IMS's dedicated GUI developers were assigned to another project, leaving Vistica with limited specialized expertise for creating the graphical front

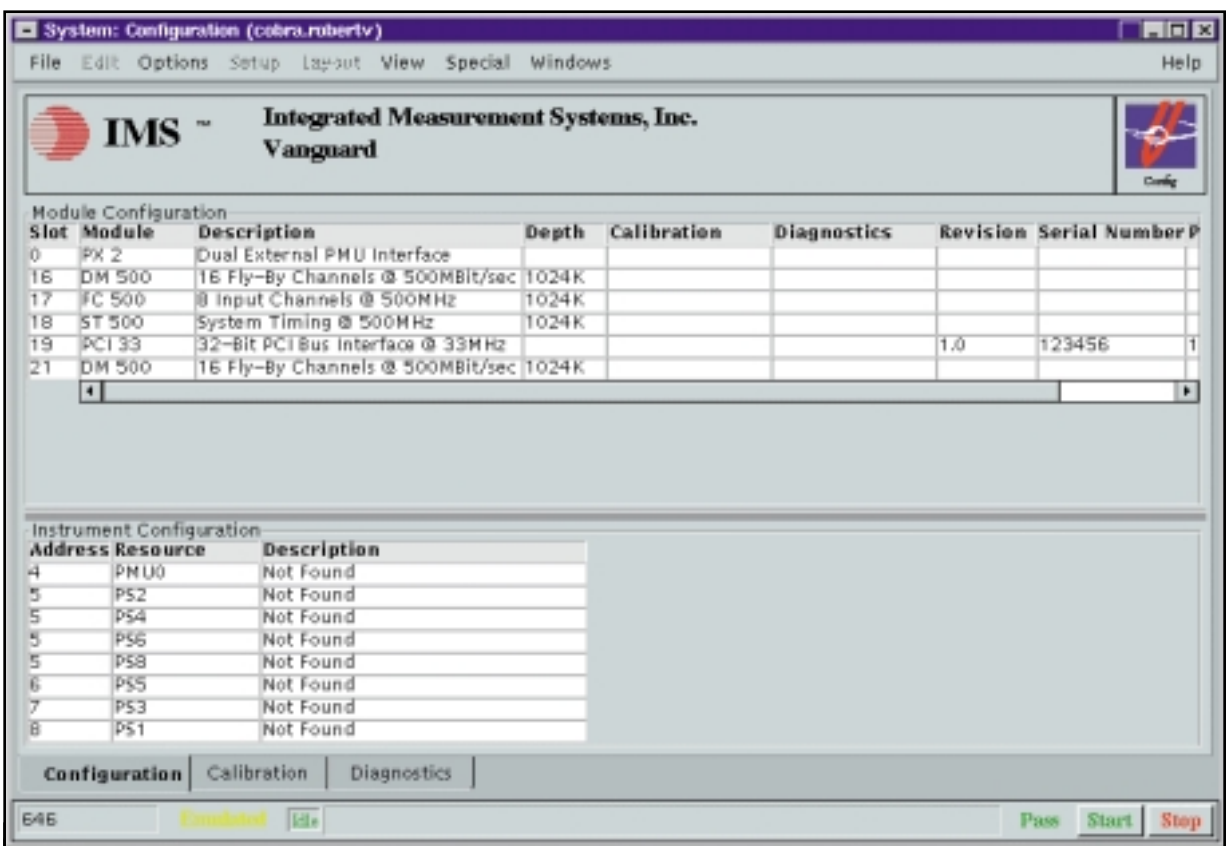


FIGURE 1 System configuration screen

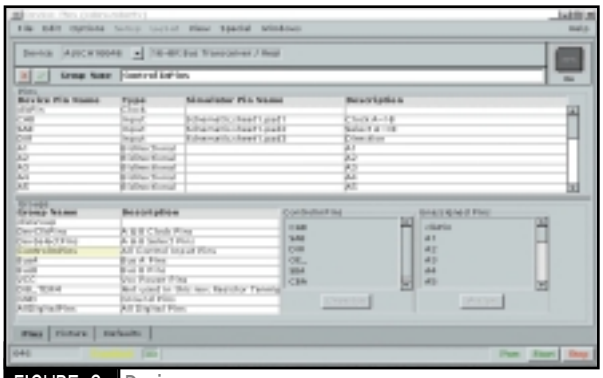


FIGURE 2 Device screen

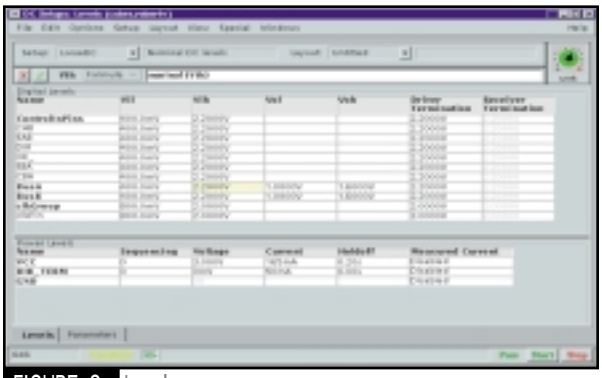


FIGURE 3 Levels screen

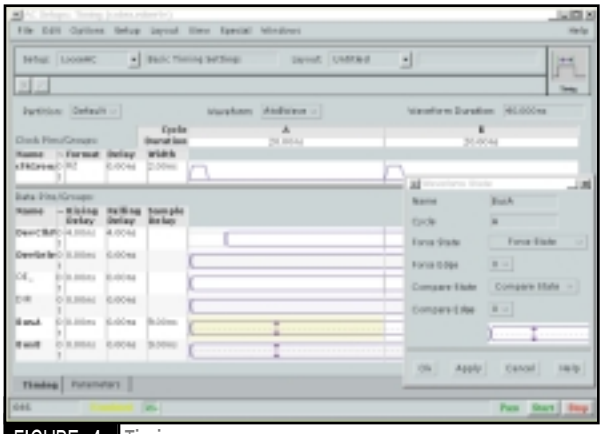


FIGURE 4 Timing screen

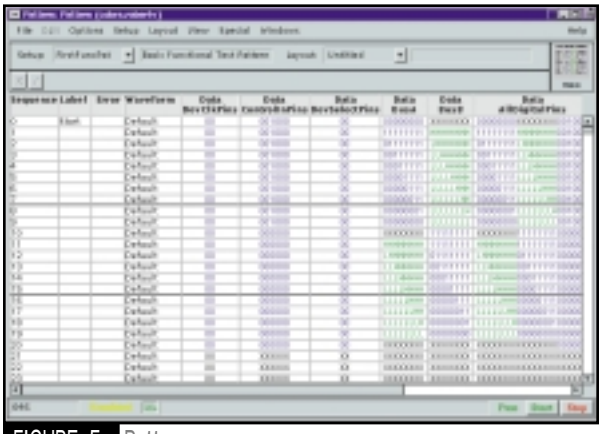


FIGURE 5 Pattern screen

end to the IC tester. With neither the time nor the developer resources to build GUI components in-house, Vistica looked to vendors of JavaBeans for ready-made, reliable GUI functionality, eventually turning to the JClass family of JavaBeans from KL Group.

"JClass was the only suite of components that offered the breadth and depth of functionality we needed. Other products just didn't come close to the feature set available in JClass."

IMS needed to create a largely table-based interface to display test results and to enable easy input of new data by users of the system. JClass LiveTable offered several capabilities that were key to the success of the project. Scalability was paramount, as Vanguard's tables would need to accommodate large quantities of data. Capable of managing tables of up to 2 billion columns by 2 billion rows of data, JClass LiveTable clearly presented no limitations in this regard. Effective data management was also important: tables would be populated with data from a variety of external data sources and had to permit live user interaction and real-time updating.

"With its robust data connectivity, LiveTable met our complex data requirements with ease," said Vistica. "We needed a solution that allowed for real-time user input, and that could handle large amounts of information."

IMS was able to prototype many of the Vanguard windows quickly using LiveTable. In addition, the move from prototype to the real software produced little throwaway code. Vanguard currently features 15 table-based windows built using JClass LiveTable, and providing a wide range of functionality:

- The system configuration screen illustrates revisions and calibrations (see Figure 1).
- The device screen lists all device pins and allows engineers to easily rename pins in a group (see Figure 2).
- The levels screen provides an easy means of setting and resetting values, giving engineers the ability to modify their tests on the fly (see Figure 3).
- Vanguard's timing screen takes advantage of LiveTable's flexible rendering model to show an actual logic diagram within a table and depicts what's happening through each cycle of the test (see Figure 4).
- At 1 million rows long, Vanguard's pattern screen, which displays every test sequence and acquired data, benefited greatly from LiveTable's scalability (see Figure 5).

The benefits of using JClass LiveTable weren't limited to optimizing Vanguard's interface functionality. From a productivity standpoint, Vistica points out that by using prebuilt components, IMS slashed two to three man-years of work from their GUI development cycle.

"The time and money savings were extremely significant: for a minimal initial investment of a couple thousand dollars, we saved close to $300K in developer costs. Building this functionality in-house would have been prohibitive. With JClass we knew we were getting reliable, well-tested functionality we could count on. The return on investment was clear," said Vistica.

Today IMS is planning to extend its use of JClass for future releases of Vanguard. Work is underway to create new windows that take fuller advantage of JClass LiveTable's powerful rendering model to add new functionality to the Vanguard interface. In addition, IMS is looking at the wide range of charting and graphing capabilities available with JClass Chart for possible integration into future projects.

"Our experiences with KL Group have been positive from day one," said Vistica. "The products are easy to use and extremely well documented. As such, we've had little need for their support, but the few times we've called on them, the responses have been timely and very helpful."

Author Bio

Sam Watts is studying computer science at the University of Waterloo, Ontario. He specializes in Java development.

sgwwatts@undergrad.math.uwaterloo.ca

Comparing Network Semantics in CORBA and Java

There's a wide variety of options for invocation and notification semantics

WRITTEN BY
JON SIEGEL

The recent issuance of an RFP for "Unreliable Multicast" in CORBA got me thinking about the many network semantics available in a combined CORBA/Java environment. There are at least five already, not counting Unreliable Multicast: Java RMI invocations; CORBA synchronous invocations; CORBA asynchronous and messaging-mode invocations; one-way notifications using the CORBA event and notification services; and the Java Messaging Service (JMS). In this column I'll review the basic characteristics of these services side by side. I'm not planning to rate them as "better" or "worse" on any scale – they're more different than better or worse, and you should choose among them based on the requirements of a particular application. The discussion will be confined to invocation semantics. While there are a lot of interesting contrasts between object activation semantics, I'll save that topic for a separate column.

Java RMI Invocations

Java RMI extends your client applications' reach, allowing them to invoke a subset of your Java objects remotely over the network. Only objects that extend java.rmi.Remote (either directly or indirectly) may be invoked via RMI, but local and remote invocations take the same form. In addition, methods must include either java.rmi.RemoteException or one of its superclasses, such as java.io.IOException. Remember, remote invocations can fail in more ways than local invocations, so it's critical to catch both system and application-specific exceptions when you return from a call.

In a Java RMI invocation there is one sender and one receiver, and the sender selects the receiver. The invocation is synchronous: the client blocks the invocation call, at least on the calling thread, until the response comes back from the remote object (or some error condition times out). The network transport is based on socket connections, with a fallback to HTTP's POST command that can penetrate through firewalls under some circumstances. There's only one quality of service (QoS), the default; you can't specify priority or time-out values for an invocation.

If you set your RMI compiler to all defaults, your RMI objects are available only from Java clients. However, if you set your flags for RMI/IDL, the compiler will generate CORBA IIOP network interfaces for your object, which will have a CORBA object reference. It'll also output the IDL corresponding to your Java object, allowing it to be invoked by CORBA clients (which may include CORBA objects acting as clients for part of their function) in the various supported programming languages.

Synchronous CORBA Invocations

This is what you get with current (2.3 and earlier) implementations of CORBA if you restrict yourself to the static invocation interface – that is, compiled IDL invoking via the client stubs. If you code to the dynamic invocation interface (DII), an interpreted version of every IDL interface that all ORBs are required to support, you have a deferred synchronous option. I won't devote space to it here because the new CORBA asynchronous method invocation provides a better alternative for most programming situations.

All CORBA objects are accessible either locally or remotely from any CORBA application that can reach them over the network, so in this respect the CORBA and Java RMI invocations are the same. There are at least two differences:

1. Certain CORBA invocations – in particular, ones with a void return value and no out or inout parameters – may be declared ONEWAY in their IDL. For these, control returns to the calling client right away, while the ORB makes a "best effort" attempt to invoke the remote object.
2. CORBA is usable from many programming languages, with standard mappings now defined for Java, C, C++, Smalltalk, COBOL, Ada, Lisp, Python and IDLscript, and nonstandardized but nevertheless useful mappings available for Objective C, Eiffel and other languages. Interfaces are written in OMG IDL rather than in a programming language; this unifies the multilanguage environment but means you can't code strictly in a single language as you can with Java RMI. At the very least you have to learn OMG IDL. I may be biased, but I think IDL is a very elegant and transparent way of defining interfaces, and compilers automatically generate client and server-side mappings for your programming language.

The mandatory protocol IIOP guarantees interoperability, while a flexible architecture allows a network to support multiple protocols. For CORBA 2.2 and prior versions there's no client-accessible API for protocol selection – or, in fact, for any network characteristic. Instead, you indicate your desired or preferred protocol in a configuration file or command-line argument at client and server start-up. As with Java, there's only one quality of service at this level, but this changes in CORBA 3.

CORBA 3 Invocation Options

The CORBA Messaging Specification, a part of CORBA 3, adds capabilities in two main areas: asynchronous method invocation and quality-of-service control, including a provision for CORBA routers that changes the network into a reliable transport. This is a very elegantly architected specification with many capabilities, so the treatment here may end up more like a list of features than a description.

The specification defines two models: a programming model and a communications model.

The programming model (Figure 1) affects only client-side interfaces. It first divides invocation space into synchronous and asynchronous. If you choose the former, you can still pick between normal invocation and the ONEWAY form mentioned earlier. If you choose the latter, you pick between callback and polling result return. Each has a different invocation API, but all are generated from your original CORBA 2-format IDL file. To make a callback invocation,

program and instantiate a callback object and insert its object reference as the first argument in your asynchronous call, which immediately returns control to your client. When the invocation completes, your callback object is called with the results. When you make a polling-mode invocation, you receive a CORBA valuetype as the return. You poll the valuetype to find out if your results are back; when it tells you they are, you invoke additional operations on it to retrieve your results.

The communications model defines CORBA routers. Each router is basically an ORB that accepts CORBA invocations, stores them and forwards them onward. Information about router locations near the server is contained in the object reference of the target; client ORBs may also be configured to know the location of routers near them. Routers may have persistent storage, enabling reliable network transport of CORBA requests and replies. (Network transmission becomes transactional, the way the best message-oriented middleware works.)

The specification goes further to define an extensive set of QoS parameters and levels, including time-outs for invocation, return or round-trip; priority levels; ordering (temporal, priority or deadline); and routing (none, forward or store_and_forward). Even though QoS is defined in the messaging specification along with asynchronous invocation semantics, its settings apply (where this make sense) to synchronous as well as asynchronous invocations.

By selecting various combinations of the programming model, communications model and QoS settings, you can vary your invocation semantics across a wide spectrum. At the simplest level you can specify basic synchronous or asynchronous invocation; at the most elaborate level you can make CORBA invocations across a network transport as robust and reliable as message-oriented middleware (MOM). Another setting allows time-independent invocations, allowing you to stage CORBA invocations to a router on your laptop while offline and having them automatically upload to the network and execute on the server the next time you connect up. (In fact, the time-independent protocol supports disconnected operation at both client and server ends!)

Real-time CORBA defines additional network QoS control. I won't give details here since space is short and real time is a specialized software area, but I will point out that it defines priority-banded connections, nonmultiplexed connections between a single client and server, and even client- and object-negotiated protocol selection. If you're writing a distributed real-time application, check this out.

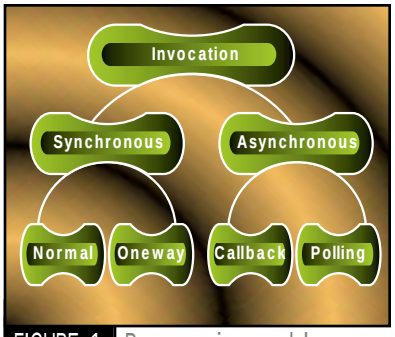


FIGURE 1 Programming model

CORBA Event Service

Java RMI and CORBA invocations invoke application-specific methods on objects written specifically for that object's interfaces in a one-to-one semantic where the invoker selects the target. Now let's turn to distributed event and eventlike services that typically invoke general, service-defined interfaces (perhaps carrying an application-specific payload stuffed into a generic type such as an any) and allow multicast or multicastlike semantics in which event suppliers and consumers subscribe to an intervening channel and may not be aware of exactly who each other is. The biggest difference between event and messaging services on the one hand and invocations on the other is that invocations use an object-oriented–type system to narrow the available operations on an object and the parameters in each operation, while event and messaging services do not. While this leaves messaging inherently more flexible, it makes invocation the more reliable choice. In large-scale systems I like to use invocation wherever I can, but move to messaging when I have to.

The CORBA Event Service, in its most widely used form, provides channels to which event suppliers and consumers connect. Push and pull semantics are supported at both supplier and consumer ends of the channel. (At the supplier end a push supplier calls a channel object while a pull supplier provides a callback object that's polled periodically by the channel for events. At the consumer end a push consumer provides a callback object to which the channel sends events as soon as they arrive, while a pull consumer polls the channel object periodically to see if any events have arrived.) Because event delivery interfaces are defined in OMG IDL, this is a distributed service intended to serve distributed (perhaps widely distributed) applications; you probably wouldn't want to use typical implementations as the basis for your desktop GUI. Event payload is an IDL any, unless you define an application-specific Typed Event, which then requires a specialized version of the service. QoS is specifically not defined, even though many service items might benefit – how timely events are delivered to push consumers, length of queue and length of time events are held on queue for pull consumers, and so forth. For this, OMG members later defined the Notification Service, which we'll look at next.

CORBA Notification Service

Defined by the Telecommunications Domain Task Force, the CORBA Notification Service adds structured events, typing and filtering, and QoS control to the Event Service, which it inherits in its entirety. (*Note:* This inheritance means only that Notification Service implementations support the entire Event Service interface set, not that they inherit a previous implementation of the Event Service. More likely, Notification Service implementations will provide Event Service functionality using code in their own implementation.)

The service defines a structured event that supports definition and application of standard filters using a constraint language. Filtering can substantially lessen the load of events on a network or a client of the service. Channels and channel maintenance are also defined. A set of discovery interfaces allows suppliers to temporarily stop supplying, or consumers to stop polling, channels that have no consumers or suppliers. QoS settings are also provided for delivery reliability, priority, various time-outs and other aspects of event delivery.

Since its definition a few years ago, the Notification Service has proved popular. Implementations are available on the software market, and its interfaces have been inherited and used in a number of subsequent OMG specifications, including the CORBA Component Model.

Java Messaging Service

The final service we'll review is a kind of hybrid. A number of third-party vendors market messaging services that provide reliable network message exchange around a central dispatcher to which all clients connect. Referred to either as *Enterprise messaging systems* or *message-oriented middleware*, these

products vary significantly in the number and type of services they build on a common peer-to-peer messaging foundation. The (JMS) defines a standard API that these vendors can layer on top of their services, allowing them to service the Java runtime environment in a way that is more or less interchangeable, depending on how much the services of one vary from those of another.

Messaging-based applications may be composed of any number of clients (of the messaging service, that is) who interact by exchanging messages. Although messages aren't invocations, JMS clients (and CORBA objects) may be programmed to take certain actions when they receive certain messages, and (according to the JMS 1.0.2 specification) a future version of EJB will include a bean that's automatically invoked by a JMS message.

Semantics vary, with some messaging services supporting point-to-point semantics, while others provide multicastlike anonymous multipoint delivery using a queuing algorithm. JMS supports both, but not interchangeably. The JMS specification points out that you won't be able to port code from one type of service to the other without substantial changes. Finally, remember that JMS works only in a Java environment.

Conclusions

Before I compare the various invocation modes and services, I want to give a few more details about the OMG specification effort currently underway to define unreliable multicast invocation semantics. *Unreliable* doesn't mean that the multicast service sometimes shows up late for work and occasionally on Monday doesn't show up at all. It signifies that the invocation semantics are "best effort," which means that message delivery isn't guaranteed, and message loss won't be detected by the transport infrastructure. (Packet loss and reordering on the other hand may be detected and corrected.) Intended primarily as a more efficient transport for the event and notification services, its underlying transport will be IP multicast rather than the connection-oriented TCP, and the IOR (Interoperable Object Reference) will specify a multicast group rather than a specific CORBA object as its target. The service also asks for definitions of object groups and methods to maintain group membership in order to support the actual multicast invocations. This will enable an event or notification service to scale to a far larger installation than currently, where each transmission of an event to a consumer is a separate CORBA invocation. Although this may not be a factor for a small number of consumers and small payloads, many installations have large numbers of consumers (I'm thinking primarily of telecommunications and other service networks here) or large payloads, such as graphics images.

In this survey we've seen that distributed applications in Java and CORBA are rapidly becoming quite capable citizens of our new networked world and that a wide variety of options are available for both invocation and notification semantics. Both have synchronous invocations, and both take advantage of reliable network transport, although CORBA does this for invocations while Java does it for messaging. CORBA has an edge in distributed event and notification delivery and works in a multilanguage environment. Of course, Java programmers can contribute objects to the CORBA environment by using RMI/IDL, which allows you to generate IDL interfaces automatically for Java objects that speak IIOP over the network while programming in pure Java. There's no corresponding magic on the client side, however, so you'll have to code your Java clients to the IDL Java language mapping to make invocations of CORBA objects on your network.

siegel@omg.org

Author Bio

Jon Siegel, director of technology transfer for the Object Management Group, has extensive experience in distributed computing, OO software development and geophysical computing. He holds a Ph.D. from Boston University.

ProtoView, JFCSuite, v2.1

Provides components that are well blended into the JFC framework

REVIEWED BY GABOR LIPTAK

AUTHOR BIO

Gabor Liptak is an independent consultant with more than 10 years of industry experience. He is currently an architect of a Java e-commerce project.

http://gliptak.homepage.com/

Protoview Development Corp.
2540 Route 130
Cranbury, NJ 08512
Phone: 800.231.8588
www.protoview.com
e-mail: info@protoview.com

Installation Requirements:
JDK versions supported: JDK1.1.5+ (with JFC) or Java2 (1.2.2+ is preferred)
Platforms: All platforms with support for the above JDK versions

Pricing: $995 online download

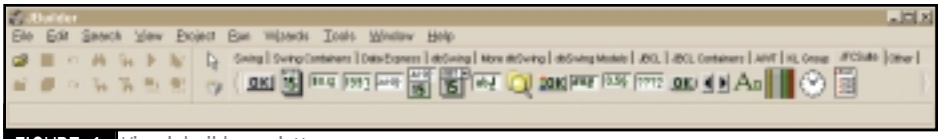


FIGURE 1 Visual builder palette

JFCSuite is a collection of visual beans based on JFC and complementing it. It fills in missing pieces in the JDK/JFC GUI libraries, namely, masked (number-only, all upper case, etc.) entry fields, date/calendar controls, various extensions (sorting) on JTable and more. All components support the Java Look and Feel (JLF) and are 100% Pure Java certified. Although the licensing is per developer, there are no runtime license fees when the library is used in commercial products. The product is also available with enterprise (priority) support bundled, which may be a good choice if the components are relied on heavily in your UI. Detailed pricing information for subscription, source code and other options are available at www.protoview.com/order/direct.asp.

The product is available in a convenient InstallShield archive format for Winx and other environments. Before installing the product, you may want to have a quick glance at the "Before You Begin" chapter of the documentation that explains which JAR files belong to which version of JFC (pv* is for Java 1.x with JFC, pvx* for Java 2). To try the product, I installed the Java 2 beans into the visual builder palette (see Figure 1) of JBuilder 3. The one thing I thought was missing when I installed the beans to the palette is a "product" JAR file (containing all beans supplied) to save some clicks when adding the beans to the palette. All icons supplied by the pv*.jar files showed up right away in JBuilder, but only several of the pvx*.jar icons did. But after I restarted JBuilder, the palette contained all the right icons.

The beans provided customizer dialogs (see Figure 2) where appropriate, although some of them felt sluggish when I saved changes.

The documentation is extensive and well prepared. It links directly to the samples, describes how to add the beans to the palettes of different Java development environments (including JBuilder) and contains suggestions on how to deploy products with the library (including information on Java Plugin). It also describes different problems encountered using the visual builder tools and workarounds in the "Design Time Notes" section. Some of the functionality isn't available in visual fashion because of the different limitations of those environments. Other platform-dependent behaviors are also described in detail in the documentation.

The components provided are well blended into the JFC framework, providing a wealth of setter/getter methods, listeners, renderers, locale settings and look-and-feel settings. The components can also be tied to each other, providing even more functionality. A good example of this is the calendar component described below, which can be used as a stand-alone or as a drop-down from another entry field for date selections.

JFCDataCalendar

JFCDataCalendar (see Figure 3) is a calendar bean that allows a date or even date ranges to be selected using a "real-life" days-of-the-month display.

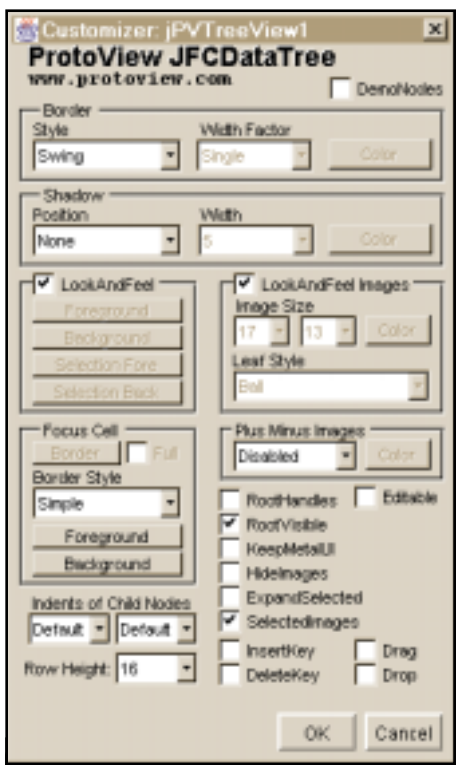


FIGURE 2 Customizer

FIGURE 3 JPVCalendar

FIGURE 4 JPVTable

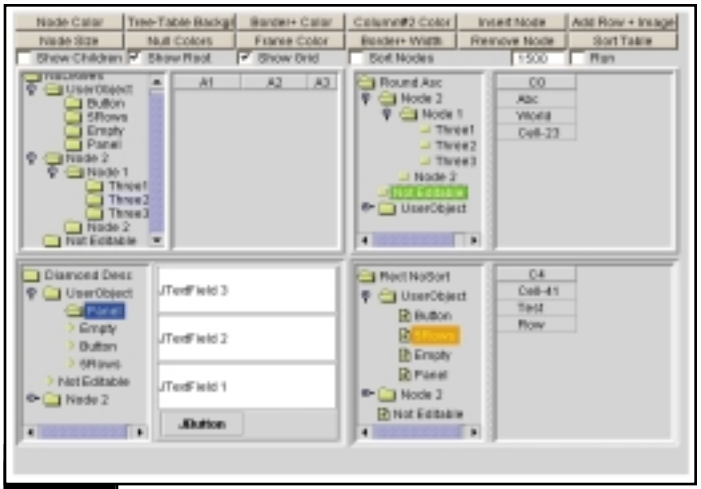


FIGURE 6 JPVDataExplorer

It can be used as a stand-alone or in conjunction with the JPVDatePlus date entry field to provide a drop-down date selector similar to the one in Quicken. The implementation even provides methods for manual placement of elements that could behave in a platform-dependent fashion. As can be seen in Figure 3, the days can be customized using images.

JFCDataTable

The JFCDataTable (see Figure 4), a drop-in replacement of the JTable component, adds sorting, printing, keyboard handling and advanced in-cell editing and formatting. The table component of this package actually subclasses from JTable, making it a true drop-in replacement.

The API has added convenience methods when compared to the JTable implementation. A "blended" scroll-and-table control is provided. Class inherits this from JScrollPane, but also implements the JTable methods.

JFCDataTree

JFCDataTree (see Figure 5) is again a true drop-in replacement for JTree, providing drag-and-drop, advanced keyboard handling, sorting, searching, node image customization and additional listeners.

A blended scroll-and-tree control is also available here.

JFCDataExplorer

JFCDataExplorer (see Figure 6) was likely inspired by the Windows Explorer UI. It ties together a JTree component with a JTable (or other components) inside a JSplitPane, allowing a display of hierarchical data structures on its left side and corresponding data on its right side. Please note that the component used to display the data on the right side, can be changed on a node-by-node basis, providing unlimited flexibility on a small screen's real estate.

Although JTable's data model can be used, a special data structure is also supplied, providing a richer information store.

JFCDataInput

JFCDataInput (JPVEdit, its subclasses and related classes; see Figure 7) offers rich data entry/validation beans. Functionality includes various masked entry fields (currency, [long] date, numeric, time) and several buttons (image, round, spin).

As for the other components above, various UI settings are offered for colors, fonts, borders, style, UI interaction timings, and so forth. The entry fields can be connected to the spin button supplied to provide a more mouse-friendly interface. I found it somewhat curious that the various masked entry fields aren't implemented using the JPVMask bean found in the package. Also, the need to have a JPVEdit superclass does seem to suggest that JFC was not as extensible as one might wish.

The JFC-style API and the extensive documentation supplied make the JFCSuite widgets a good choice if your UI needs the functionality described above.

At press time, ProtoView was scheduled to release version 3.0 to the JFCSuite. This version will include a DayView component as well as advanced n-tier data models for data binding.

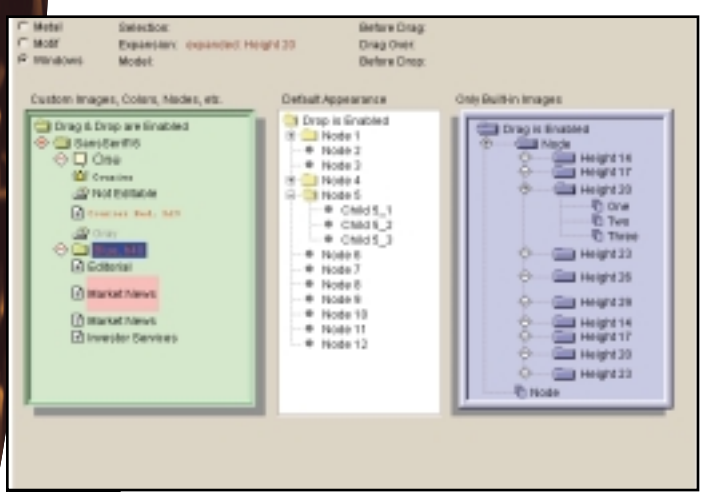
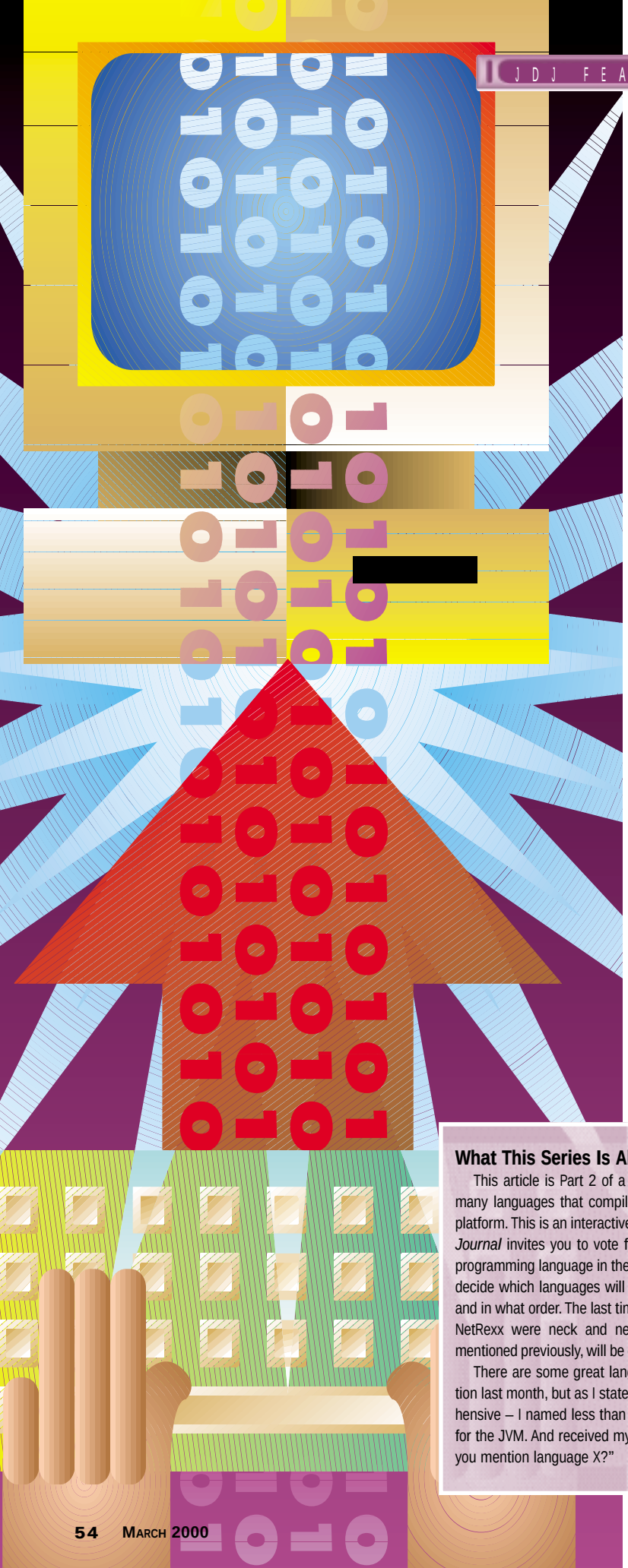


FIGURE 5 JPVTree

FIGURE 7 JPVEdit



JDJ FEATURE

Python Programming in the JVM

Increase your productivity by putting Python in your toolbox

WRITTEN BY RICK HIGHTOWER

What This Series Is About

This article is Part 2 of a series that discusses the many languages that compile and/or run on the Java platform. This is an interactive series. *Java Developer's Journal* invites you to vote for your favorite non-Java programming language in the *JDJ* Forum. Your vote will decide which languages will be covered by the series, and in what order. The last time I checked, JPython and NetRexx were neck and neck. NetRexx, though not mentioned previously, will be covered in the next article.

There are some great languages that I didn't mention last month, but as I stated, the list wasn't comprehensive – I named less than 10% of all the languages for the JVM. And received my fair share of "Why didn't you mention language X?"

Most of the languages covered by this series are scripting languages that are dynamic, interpreted and easy to program. For this article and the ones that follow, Java will be presented as the system language and the higher-level language will be presented as the "glue" language. Thus you define frameworks, libraries and components in Java and glue them together to make applications. This was described in detail in *JDJ*, Vol. 5, issue 2.

The series will focus on topics such as other languages for the JVM; integrating Java with mainstream scripting languages like Perl and Python; special-purpose languages (rules, etc.); creating JavaServer Pages (JSP) in JavaScript, WebI and Python; SWIG; open-source Java initiatives; COM/DCOM from pure Java; and CORBA to legacy integration.

JPython is the 100% Pure Java version of Python and is freely available – source code and all. An extremely dynamic, object-oriented language, JPython is in its second major release – JPython 1.1. Since JPython is the same syntax and language as Python, I'll use the terms interchangeably for the rest of this article.

You've heard Java called a dynamic, object-oriented language. Well, JPython is more dynamic and more object-oriented than Java. In Python, unlike Java, everything is an object – classes, methods, namespaces are objects. Also, Python doesn't have any primitive types, and it supports multiple inheritance. In many ways Python is closer to Smalltalk than to Java – syntactically, however, Python is closer to Java than to Smalltalk.

This isn't a case of my-language-can-beat-up-your-language syndrome. JPython doesn't replace Java; it augments it. Java and JPython have complementary roles – sometimes they overlap.

JPython facilitates the following:

- ***Embedded scripting language:*** You can add JPython to your application to enable those pesky, demanding end users to extend your applications through scripts. Thus your end users can extend your application to add functionality that only a domain expert could dream of.
- ***Interactive experimentation:*** JPython, like many scripting languages, provides an interactive interpreter. I use this to try out new APIs and for prototyping. Also, this is a great debugging tool – you can execute methods in any order, not in the normal sequence. Since the syntax is close to Java, it's easy to prototype.
- ***Rapid application development:*** Python programs are shorter than the equivalent Java programs, as I'll show in the Rosetta stone examples.

Python is a lot easier to learn than Java. A novice programmer can learn enough Python in half a day (or sometimes in a few hours) to write effective scripts. In addition to being good for programming-in-the-small, you can use it for larger programs. Python has advance namespace management that makes programming-in-the-large feasible – many scripting languages don't. For example, Python has packages similar to Java's.

Python's ease of use is on a par with Visual Basic. Some say Python even surpasses Visual Basic because the syntax is more consistent and designed. However, unlike Visual Basic, Python is truly object-oriented. Others say that Python closely resembles a nonverbose version of Pascal or C++. The ease-of-use syntax is good, and the dynamic capabilities are extremely powerful. Put a Python in your toolbox.

Rosetta Stone

For comparison, each JPython sample application will have a corresponding Java implementation. This article covers the following sample applications.

- *A simple GUI application*
- *A simple statistics application*
- *Embedding the script into an application (if applicable)*
- *A simple example parsing text*

Before we go into the first example, let's cover the basics of JPython. If you've installed JPython, please follow along in the interactive interpreter.

A Simple Class

Python has classes. Listing 1 is a sample Python class; the Java equivalent appears in Listing 2. (The remainder of the Listings, through Listing 16, can be downloaded at www.javaDevelopersJournal.com.)

Notice that the use of *self* is similar to the use of *this* in a Java class – *self* is the reference to the instance. The first argument to each method is a reference to *self*. Also note that there's no separate declaration for member variables, that is, they're declared when they're assigned a value. (You can declare class variables as well as instance variables.) The _str_method is a special method that acts like the toString method in Java. Compare the Python Employee class to the roughly equivalent Java class in Listing 2.

To create an instance of an Employee and print it to the screen you'd do the following:

```
print Employee()
```

The equivalent Java statement would be:

```
System.out.println(new Employee());
```

Next we create two instances of Employee called joe and ron, and print those employees to the console. We print joe, who is ron's manager, by invoking the getManager method of ron – first in JPython, then in Java

PYTHON

```
joe = Employee("Joe", "Batista", 100)
ron = Employee(manager=joe, id=101, lname="Furgeson", fname="Ron")
print ron
print ron.getManager()
```

JAVA

```
Employee joe = new Employee("Joe", "Batista", 100, null, 1);
Employee ron = new Employee("Ron", "Furgeson", 101, joe, 1);
System.out.println(ron);
System.out.println(ron.getManager());
```

As I said, the syntax is similar. One feature that JPython has is named *arguments and default values*. Notice that when the ron instance is created, the arguments are called out of order. If you've programmed in Visual Basic or VBScript, this concept should be familiar. If not, think of it this way: you can call methods as you normally do with Java, or you can pass name, value pairs to the method as shown above. This feature can save some coding effort, not to mention some headaches. Have you ever had several versions of the same method? And you just wanted to have different default values? Every default value is another overloaded method. It can get messy.

A good example of using named arguments is the GridBag utility class that the JPython distribution provides. This utility helps you manage the infamous GridBagLayout. I've created something similar in Java that used overloaded methods to create GridBag constraints. I was amazed how short the GridBag utility was in Python. (It's in the pawt package, if anyone wants to check it out.)

A Simple GUI

Now that we've created a simple class, we'll create a simple GUI. I admit that the class and the GUI are nonsensical – the idea is to demonstrate and compare the language to Java.

If you have JPython installed, let's pretend that we're prototyping this GUI. Fire up the interactive interpreter by typing jpython as the system prompt. (This assumes that you've downloaded, installed and put JPython home directory on your Path. Follow the install instruction at www.jpython.org.)

Import the JFrame from the javax.swing package.

```
>>> from javax.swing import JFrame
```

Create an instance of the frame, set its size to 200 by 200, then make it visible.

```
>>> frame = JFrame("My Prototype", visible=1, size=(200,200))
```

You probably weren't expecting this to be only one line of code. In JPython any bean property of a class can be set during the call to the constructor using named arguments. By *bean property* I mean a property as defined by a getter and a setter method, that is, the bean "design pattern" for properties.

Design

tdesign.com

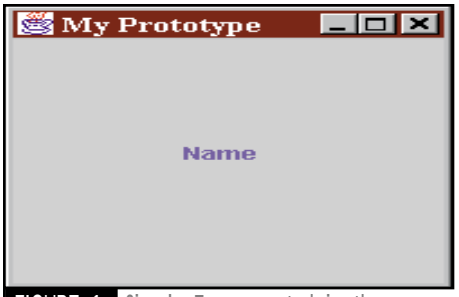


FIGURE 1 Simple Form created in the interactive interpreter

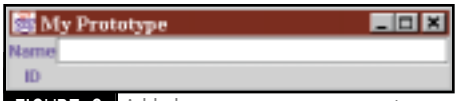


FIGURE 2 Added some more components, packed the frame and arranged components

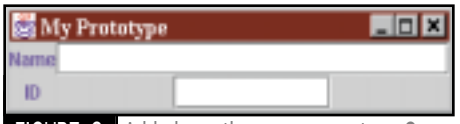


FIGURE 3 Added another component....Oops, made a mistake. Better fix it.

FIGURE 4 Fixed layout. Looks good!

FIGURE 5 Added okay button and event handler

At this point our frame is pretty boring. A stupid-looking gray box. Let's add some components to our stupid-looking gray box. We need to add some labels, text fields and an okay button. As we develop this GUI application I'll point out some of the features of JPython. As it's a nonsensical demo, we aren't going to meet any GUI style guidelines. First we need to import a few classes from javax.swing.

```
>>> from javax.swing import JButton, JTextField, JLabel, JPanel
```

Notice that we didn't use the * syntax. For example, we could have said "from javax.swing import * as you'd do in Java. But that would have imported every class into our namespace – in Python that would be considered bad style. In Python you can view and manipulate the namespace. For example, to see all of the variables in the current namespace, you can do the following:

```
>>> dir()
['JButton', 'JFrame', 'JLabel', 'JTextField',  '__name__', 'frame']
```

Thus, if we imported *, we'd have a lot of classes in our namespace, and that would be less than ideal.

First create a pane. (Notice how the frame's contentPane property is handled; in Java you'd have to call frame.getContentPane() to get the contentPane. Bean properties are treated like instance variables.)

```
>>> pane = JPanel()
>>> frame.contentPane.add(pane)
javax.swing.JPanel[,0,0,0x0,invalid,layout=java.awt.FlowLayout,align-
mentX=null,alignmentY=null,border=,flags=34,maximumSize=,minimum-
Size=,preferredSize=,default
Layout=java.awt.FlowLayout[hgap=5,vgap=5,align=center]]
```

For this example we'll use the GridBag utility class that's provided with JPython. GridBag makes using GridBagLayout easy. The amazing thing about GridBag is how few lines of code it took to write it – again, check it out.

First we import the GridBag helper class, then create an instance of GridBag and associate it with the Pane. Please follow along in the interactive interpreter.

```
>>> from pawt import GridBag
>>> bag = GridBag(pane)
```

Now add the first component to the grid bag – a JLabel. This will use all of the default values of the GridBagConstraints (see Figure 1).

```
>>> bag.add(JLabel("Name"))
>>> frame.validate()
```

Now add another JLabel. This time we add the label on the second row of the grid.

```
>>> bag.add(JLabel("ID"), gridy=1)
>>> frame.validate()
```

Now add a text field on the first row in the second column. Then pack the frame (see Figure 2).

```
>>> name = JTextField(25)
>>> bag.add(name, gridx=1, weightx=80.0)
>>> frame.pack()
```

Now add a second text field for the employee ID, this time to the right on the second row. Then pack the frame (see Figure 3).

```
>>> id = JTextField(10)
>>> bag.add(id, gridx=1, gridy=1, weightx=80.0)
>>> frame.pack()
```

As you can see, this isn't what we want. The text field components are centered and look quite silly. I forgot to align the text field to the left in their cells (not really – I forgot on purpose).

Let's remove the components and add them again with the proper alignment. Hopefully, you'll see how useful it is to be able to experiment with the layout in the interactive interpreter (see Figure 4).

Remove the ID and name.

```
>>> pane.remove(id)
>>> pane.remove(name)
```

Now re-add the ID and name with the proper alignment.

```
>>> bag.add(name, gridx=1, weightx=80.00, anchor='WEST')
>>> bag.add(id, gridx=1, gridy=1, weightx=80.0, anchor='WEST')
>>> frame.pack()
```

The above demonstrates the interactive, experimental environment. You can explore cause and effect without the normal recompile, retest environment.

Bean events are handled easily in JPython. As with bean properties, JPython adds features to make event handling easier. First, JPython uses introspection and reflection to make event properties. Event properties equate to the names of the methods in the event listener interface for a given method using the event "design pattern" for JavaBeans.

You can assign an event property a function or method. To demonstrate this, let's set up an okay button. When the okay button gets clicked, this prototype application will print out the employee's name and ID.

First create and add a button to our GUI (see Figure 5).

```
>>> okay = JButton("Okay")
>>> bag.add(okay, gridx=1, gridy=2, anchor='CENTER')
>>> frame.pack()
```

Next create a function. The function prints out the value of the name and ID text.

```
>>> def handleOkay(event):
...      print "Name " + name.text
```

```
...     print "ID   " + id.text
...
>>> okay.actionPerformed=handleOkay
```

Enter some text in the Name and ID field and hit the okay button. This is a simple session, creating a simple GUI. For those of you who followed along with JPython, let me know what you think of JPython. I like it and use it often.

Rosetta Stone GUI

The foregoing was to show the interactive interpreter that comes with JPython. Now let's create a GUI based on the one we created above in both JPython and Java. In subsequent articles we'll write the same GUI in NetRexx, JavaScript, BeanShell, and others, enabling you to compare the JPython example with an example in NetRexx.

Listing 3 shows the employee form that we prototyped in the interactive interpreter. Listing 4 shows the employee form in Java. The Java version is 2,139 characters while the JPython version is 1,290 characters; thus the Java version is 66% larger.

Rosetta Stone Statistics

Just to highlight how well the language can do simple, common things, we'll create a simple application that calculates statistics for house prices in a neighborhood. Thus we'll create a program that, given a list of numbers, finds the averages (mean, mode, median) and range. We'll list each function's code and then break it down and describe the function line by line.

IMPLEMENTING GETRANGE

Since getRange is the easiest, we'll do it first. Essentially, we want a function that returns the minimum and maximum values in a list, and the range of values in the list (see Listing 5).

FIRST TRY IMPLEMENTING GETRANGE

Range iterates through a set of numbers passed to it and calculates the minimum and maximum values. When it's done, it returns the min, max and the range in a tuple.

Let's break getRange down bit by bit. First getRange declares two variables called *min* and *max*. The min is to hold the minimum value. The max is to hold the maximum value.

```
min = 300000000
max = -300000000
```

The min variable refers to a very large number so that the first item extracted from the list will be less than the large number and get assigned to the min value. The max value contains a very negative number so that the first item that gets extracted will be more than the large negative value and get assigned to min variable.

Technical Note:

The foregoing example will work only if the numbers passed to it are in the range of min and max.

A better way to implement this code would have been to use the following:

```
from java.lang import Double
```

or:

```
from java.lang import Integer
```

and then:

```
min = Double.MAX_VALUE
max = Double.MIN_VALUE
```

or:

```
min = Integer.MAX_VALUE
max = Integer.MIN_VALUE
```

This would make the function work well with IntTypes or DoubleTypes, but what about LongTypes? Well, there's a more Python way of doing things, which will be explained shortly.

Next, to figure the minimum and maximum number, the getRange function iterates through the nums sequence.

```
for item in nums:
  if (item > max): max = item
  if (item < min): min = item
```

Notice the JPython for loop iterates through a sequence. A sequence is like a cross between a Java Array and a Java Vector – well, not exactly.

The expression item > max determines if the item's value is greater then the value of max. If it is, it's assigned to the value of the item. This, so far, is a lot like Java.

When the loop stops iterating the values, the getRange function returns the min, max and range (max - min) as:

```
return (min, max, max-min)
```

This may be odd. Essentially, it appears that we're returning three values. Actually, we're returning a tuple of values. A *tuple* is an immutable sequence.

Well, that was easy enough. If you read the technical note, you know this approach has some flaws.

We're getting a variable called nums. The *nums* variable is a sequence. In Python, sequences have intrinsic operations (built-in functions) for finding the min and max values in a list. Therefore, we should have used the built-in function.

The Python Way of getRange

This is an improvement of getRange, because it's a lot shorter and it can work with all numeric types, longs, floats and doubles at their maximum range of precision (see Listing 6).

There's no more for loop, no more figuring out what the minimum or maximum variable should be initialized to. The built-in intrinsic functions in Python are very useful. Now this will work with longs, integers and floats. (Read the technical note under the first implementation for a description of the getRange problem).

IMPLEMENTING GETMEAN

The getMean function figures out the mean of a sequence of numbers. It iterates through the list, adds all the values together and stores them in sum. It then figures the mean by dividing the sum divided by the length of the sequence of numbers. The getMean sample uses an argument called *sample* to determine if this is a sample mean or a population mean (see Listing 7).

This example shows example use of:

- *For* loop
- *If* and *else* statements
- The built-in function called *len*

Let's break down the getMean function step by step.

First create a variable called *sum* that holds the sum.

```
sum = 0.0
```

Then iterate through the nums sequence accumulating the value of the item *x*.

```
 for x in nums:
  sum = sum + x
```

Next we check to see if this is a sample mean. If it is, we figure the average by dividing the sum by the number of items in nums less one, else we divide the sum by the number of items in the nums sequence.

```
 if(sample):
  average = sum / (len(nums)-1)

  # Else it is a population mean
 else:
  average = sum / len(nums)
```

Last, we return the average.

```
return average
```

The foregoing is not much different from what you'd do in Java. You could say that it's very similar.

The Python Way of getMean

There's another way to calculate the average. It's quite different from what you may be used to.

Examine the following code.

```
def getMean (nums, sample):
 sum = reduce(lambda a, b: a+b, nums)

 if sample: average = sum / (len(nums)-1)
 else: average = sum / len(nums)

 return average
```

This approach uses a built-in function called *reduce*, which takes two arguments – a function object and a sequence object. The former is specified with the lambda keyword, which defines an anonymous function, that is, a function without a name.

Thus:

```
lambda a, b: a+b
```

is equivalent to

```
 def add(a,b):
  return a + b
```

The reduce function applies the function argument to two items in the sequence argument cumulatively from left to right, which reduces the sequence to a single value that in our case is a sum.

Thus the foregoing method is equivalent to the former getMean method, but a lot shorter. There are other built-in functions like reduce that provide functional programming features to Python.

IMPLEMENTING GETMODE

The getMode function finds the value that repeats the most. *(Note:* This isn't a complete implementation of mode – it would work only with discrete values.) The first thing this function does is duplicate the sequence, because it's going to modify it. Then we iterate through the items in the nums sequence, and count the numbers occurrences of the current items (we use the built-in sequence method count). Once we count an item, we remove it from the duplicated sequence (see Listing 8).

This example shows example use of:

- *For* loop
- *While* loop
- *If* statement
- Built-in intrinsic operation *count*
- Also uses a nested *for* loop

Because of space considerations, we won't cover getMode step by step.

IMPLEMENTING GETMEDIAN

The getMedian function finds the middle-most value once the sequence is sorted (see Listing 9).

This example shows example use of:

- The modulus operator (%)
- *If* and *else* statements
- Built-in intrinsic operation *sort*

Also because of space considerations we won't cover the getMedian function step by step. Let's highlight some of the getMedian functionality. First we duplicate the nums sequence. Then we sort the duplicate (the duplicate is named seq):

```
 seq.sort()
```

Next we get the length of the sequence and check to see if it's an even number with the expression length % 2. (Remember that the modulus operator – % – returns the remainder, so if length is even, the expression length % 2 will return a 0.) If the length is even, we calculate the median by adding the two most central numbers and figuring their average.

```
 length = len(seq)

 if ( ( length % 2) == 0):
  index = length / 2
  median = (seq[index-1] + seq[index]) /2.0
```

If the length is odd, we grab the middle value:

```
 else:
  index = (length / 2)
  median = seq[index]
```

Last, we return the median.

```
 return median
```

IMPLEMENTING REPORTSTATISTICS

Last, we want to print out the statistics we collected. The reportStatistics calls all of the functions; we implement and store their return values in two dictionaries called *averages* and *ranges*. It then puts the two dictionaries in another dictionary called *report*. It returns report (see Listing 10).

This example shows example use of:

- getMean, getMode, getRange, getMedian
- Nested dictionaries

Let's cover the reportStatistics function step by step.

Get the averages – namely, the mean, median and mode. Use the getMean, getMedian and getMode functions that we defined. Note that "mean":getMedian defines a key value pair.

```
 averages = {
  "mean":getMean(nums,0),
  "median":getMedian(nums),
  "mode":getMode(nums)
  }
```

Get the range parameters – namely, the min, max and range – from the getRange function. Use the range[0], range[1] and range[2] items in the sequence returned from the function getRange. Note that "min":range[0] defines a key value pair in the ranges dictionary. Unlike Java, Python has

built-in support for collections. Thus you can specify a dictionary, which is like a java.util.Hashtable, with a literal.

```
  # get range
 range = getRange(nums)

  # put ranges in a dictionary
 ranges = {
  "min":range[0],
  "max":range[1],
  "range":range[2]
  }
```

Now define a dictionary called report that contains the averages and ranges dictionary:

```
 report = {
  "averages": averages,
  "ranges": ranges
  }
```

Last, let's return the report dictionary

```
 return report
```

USING REPORTSTATISTICS

The runReport module uses the reportStatistics to get the report dictionary that it uses to print out the report (see Listing 11).

This example shows example use of:

- The string format operator (%)
- the %f format directive
- Using nested dictionaries
- Using a dictionary with the format operator

The string formatter (%) operator is like the printf function in C except that the % operator can work with dictionaries of data (see Listing 12). I've used the string formatter to generate Java code. It's one of my favorite Python features and really makes text reporting easy.

As you can see, there are a lot of built-in functions and built-in collection types that make easy tasks easier and hard tasks possible. Now compare this to the Java version of this application in Listing 13. Notice how much shorter the Python version is.

Rosetta Stone String Parsing

This example will continue where the other one left off. We'll add reading the house prices from a file. The file will consist of a comma-delimited list of house prices.

For example, the file will contain:

```
100000,100000,120000,150000,170000,170000,80000,50000
```

The Python code to read this file in would be as follows:

Open the file.

```
>>> file = open("data.txt")
```

Read in the file data.

```
>>> data = file.read()
```

Import the split function to parse the data.

```
>>> from string import split
>>> housePrices = split(data, ",")
```

For demonstration purposes show that the split function split the data string into a list of strings.

```
>>> housePrices
['100000', '100000', '120000', '150000', '170000', '170000',
'80000', '50000\n'
```

Convert the housePrices list from a list of strings to a list of floating point values.

```
>>> housePrices = map(float, housePrices)
```

Show that the list is now a list of floating point values.

```
>>> housePrices
[100000.0, 100000.0, 120000.0, 150000.0, 170000.0, 170000.0,
80000.0, 50000.0]
```

Listing 14 is the listing for the foregoing prototype.

Compare the JPython listing for runReport2 (Listing 14) and the Java listing (Listing 15) for runReport2. As before, the JPython version is much shorter.

Rosetta Stone Embedding JPython in Java

A good example of embedding JPython in Java ships with the standard JPython distribution (see Listing 16). I've added comments to the example.

As you can see, it's relatively easy to embed JPython into a Java program. The example doesn't do the subject justice. In my book, *Programming the Java APIs with JPython*, there's an example that dynamically loads JPython servlets from a Java servlet using the embedding technique shown.

Vote for Your Favorite

A lot of 100% pure programming languages work in the JVM, and ***JDJ*** wants your opinion of which are the best, and why. An example list follows: JPython, NetRexx, Rhino, Instant Basic, Jacl, BeanShell, Pnuts, Bistro, Kawa (Lisp/Scheme like).

Scorecard

How does JPython score? The table below shows my opinion – 90%. Drop by the ***JDJ*** Forum and put in your own two cents' worth!

AUTHOR'S SCORECARD FOR JPYTHON

Ease of use	10
Embeddability	10
Resemblance to parent language	10
Unique features	10
String parsing	10
Productivity	10
Working well with Java classes	10
Development environment/debugging	2

Let's drill down a bit on the criteria I rated.

- ***Ease of use:*** Python, written by Guido Van Rossum, was based on ABC, which was developed to make programming easy for beginners. Python is easy to use and easy to learn. In Virginia they're using Python as a first programming language to teach high school students how to program. Score 10 of 10
- ***Embeddability:*** Python is easy to embed. Score 10 of 10
- ***Resemblance to parent language:*** JPython strives to be syntactically identical to Python. Score 10 of 10
- ***Unique features:*** Python has some of the best features of Smalltalk, Scheme, Icon and Java. Score 10 of 10

—Continued on page 96

Listing 1: Python Employee Class

```
…
class Employee:
 def __init__(self, fname="John", lname="Doe", id=1, manag-
er=None, dept=1):
  self.__firstName         =        fname
  self.__lastName =        lname
  self.__id                =        id
  self.__manager           =        manager
  self.__dept              =        dept

 def getManager(self):
  return self.__manager

 def __str__(self):
  values = self.__lastName,
self.__firstName, self.__id
  return join(values,',')
```

Listing 2: Java Employee Class

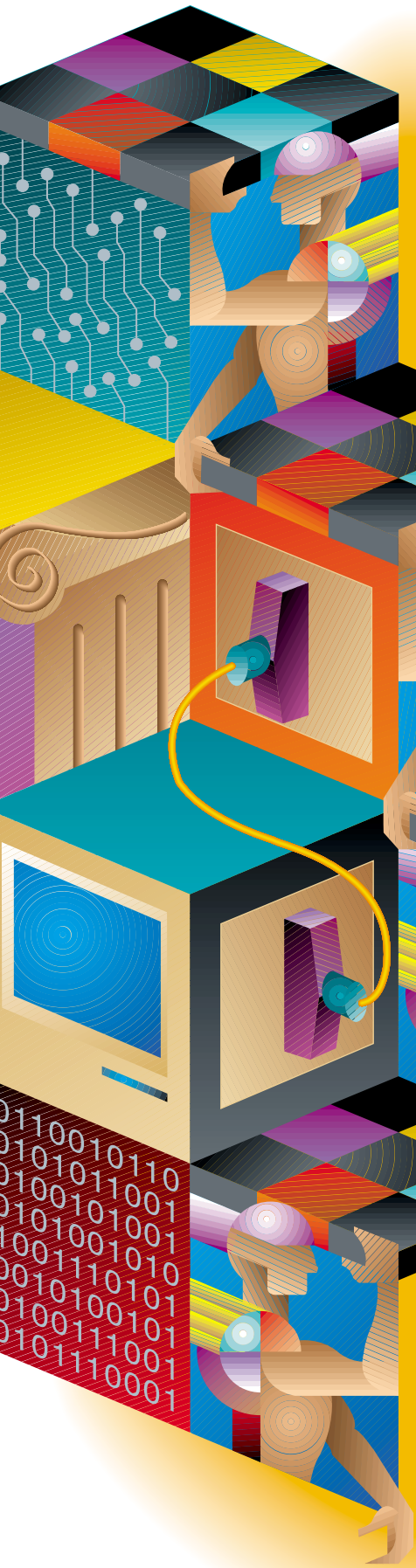
```
public class Employee{
 private String firstName, lastName;
 private int id, dept;
 private Employee manager;


 public Employee(){
  firstName = "John";
  lastName = "Doe";
  id = 1;
  manager=null;
  dept=1;
 }

 public Employee(String fname, String lname, int id, Employee
manager, int dept){
  firstName       =        fname;
  lastName        =        lname;
  this.id                  =        id;
  this.manager    =        manager;
  this.dept       =        dept;
 }

 public Employee getManager(){
  return manager;
 }

 public String toString(){
  StringBuffer buf = new StringBuffer();
  buf.append(lastName+',');
  buf.append(firstName+',');
  buf.append(""+id);
  return buf.toString();
 }
 …
 …
}
```

Listing 3: JPython Employee Form

```
from javax.swing import JFrame, JButton, JTextField, JLabel,
JPanel
from string import split
from pawt import GridBag
from Employee import Employee


class EmployeeForm(JFrame):
 def __init__(self):
  JFrame.__init__(self, "Employee Form")
  pane = JPanel()
  self.contentPane.add(pane)
  bag = GridBag(pane)


   #Create a name, and id text field.
  self.__name = JTextField(25)
  self.__id = JTextField(10)

   #Create and add a "Name" and "ID" label.
  name = JLabel("Name", labelFor=self.__name, displayedM-
nemonic=ord('N'))
  bag.add(name)
  id = JLabel("ID", labelFor=self.__id,
displayedMnemonic=ord('I'))
  bag.add(id, gridy=1)

   # Add the name and ID text field to the form.
  bag.add(self.__name, gridx=1, weightx=80.00, anchor='WEST')
  bag.add(self.__id, gridx=1, gridy=1, anchor='WEST')


   #Create an okay button, add it, and set up its event han-
dler.
  okay = JButton("Okay", mnemonic=ord('O'))
  bag.add(okay, gridx=1, gridy=2, anchor='EAST')
  okay.actionPerformed=self.handleOkay

  self.visible=1
  self.pack()

 def handleOkay(self, event):
  fname, lname = split(self.__name.text, " ")
  id = int(self.__id.text)
  employee = Employee(fname, lname, id)
  print employee


if __name__=="__main__":EmployeeForm()
```

Listing 4 :Java EmployeeForm

```
import javax.swing.*;
import java.awt.GridBagLayout;
import java.awt.GridBagConstraints;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import employee.Employee;


public class EmployeeForm extends JFrame{
 private JTextField name;
 private JTextField id;

 public EmployeeForm(){
  super("Employee Form");
  JPanel pane = new JPanel();
  getContentPane().add(pane);

  pane.setLayout(new GridBagLayout());


   // Create a name, and id text field.
  name = new JTextField(25);
  id = new JTextField(10);

   // Create and add a "Name" and "ID" label.
  JLabel nameLabel = new JLabel("Name");
  nameLabel.setLabelFor(name);
  nameLabel.setDisplayedMnemonic('N');
  GridBagConstraints constraint = new GridBagConstraints();
  pane.add(nameLabel, constraint);

  JLabel idLabel = new JLabel("ID");
  idLabel.setLabelFor(id);
  idLabel.setDisplayedMnemonic('I');
  constraint.gridy=1;
  pane.add(idLabel, constraint);

   // Add the name and ID text field to the form.
  constraint.gridy=0; constraint.gridx=1;
  constraint.weightx=80.00;
  constraint.anchor=GridBagConstraints.WEST;
  pane.add(name, constraint);
  constraint.gridy=1;
  pane.add(id, constraint);
```



WRITTEN BY A.V.B. SUBRAHMANYAM

Java Servlets: Design Practices

Part 2

AN ANALYSIS OF DESIGN PRACTICES USED BY SERVLET DEVELOPERS IN WEB APPLICATIONS

The Java servlet API specifies a very lightweight framework for developing Web applications. Although servlet technology is just one of the building blocks in the J2EE architecture, developers often use servlets to build full-fledged Web applications. Today several vendors and organizations provide servers and containers that implement the servlet API. For an overview of the servlet programming model, and some of the advanced features of Java servlets, refer to Part 1 of this article (***JDJ***, Vol. 5, issue 2).

As a lightweight framework, the servlet API doesn't impose a very strict programming model. Specifically, the API specification leaves certain design decisions to the developers. The servlet specification also added and withdrew (deprecated) certain features over its evolution. Unfortunately, though such deprecation was necessary to bring robustness to the servlet model, such changes sometimes cultivated incorrect programming practices. In addition, container vendors follow different models for implementing the API specification. In such a scenario it's essential for servlet programmers to be careful while making assumptions regarding those aspects that aren't covered by the servlet specification.

In this article I'd like to conduct an analysis of some of the design practices that servlet developers often undertake while developing Web applications. I draw some of the inputs for this article from the archives of the Servlet Interest Mailing List. *Note:* Some of the practices may not affect small-scale Web applications. The focus of my article is on servlets for large-scale Web applications that are required to be portable and scalable. While discussing these practices, I assume the containers will provide advanced features such as clustering, failover, and more.

Explicit Servlet Instantiation

This is a novice practice. Experienced programmers would probably snub me for mentioning this because they never do it. Nonetheless, allow me to elaborate on why you shouldn't instantiate servlets explicitly.

In brief, the hosting servlet container creates servlet instances. Instances created explicitly by applications can't receive HTTP requests from clients.

A servlet container creates servlet instances in response to HTTP requests. As discussed in Part 1 of this article, the purpose of a container is to receive HTTP requests, construct or locate a servlet instance, construct the environment for the servlet instance (a ServletContext), construct the request and response objects (the HttpServletRequest and HttpServletResponse objects), and delegate the incoming request to the servlet instance.

The key point here is that it's the servlet container's responsibility (and prerogative) to create servlet instances and delegate requests. Both the threading model that the developer follows and the instantiation model implemented by servlet containers dictate how and when servlet instances are created.

Applied Reasoning

www.appliedreasoning.com

If a servlet implements the SingleThreadModel interface, the servlet container delegates each concurrent request to a different servlet instance or serializes incoming requests so that a single instance handles all the incoming requests, one after the other. If your LoginServlet implements the SingleThreadModel interface and there are five concurrent requests to this servlet, there are two possibilities. The first is that the container will delegate these five requests to five different instances of the LoginServlet. Alternatively, each of these login requests will be handled in series by just one instance. Once all the requests are served, the container may hold the instance(s) in a pool and reuse them for future requests.

For servlets that don't implement the SingleThreadModel interface, the instantiation policy depends on the container implementation. While some containers maintain a pool of instances, with each instance handling requests in different threads, some other containers delegate all requests in different threads to just one instance. However, the new Java Servlet API 2.2 specification mandates that a container must maintain one instance per servlet per Java Virtual Machine in an application.

Servlet instances are created and destroyed according to the policies . There's nothing that actually prevents you from creating a servlet instance as long as the LoginServlet class is available in the container's class path. Such an instance will be independent of the host servlet container. The container can't know about it. It can't manage the life cycle of such an instance. You're probably loading the servlet container in some unwanted way.

What's Wrong with Instance Variables?

What's wrong with instance variables in servlets? After all, servlets are normal Java objects. Well, yes, they are normal Java objects. But the servlet container manages them and that makes all the difference.

There are at least two reasons why a servlet developer would want to have instance variables in a servlet. One is to store request-specific state (e.g., state pertaining to a user session including any conversational state); the second is to store state that's common to several requests (e.g., application state).

In either case servlet instance variables are not the best solution for handling state. Let me rule it out completely before we discuss better solutions provided by the specification.

In addition to the container's instantiation model and the servlet's threading model discussed above, two more issues are to be considered here:

1. ***Clustering and load balancing:*** As discussed in Part 1, if your container provides clustering facilities, you can mark your Web applications as distributable. In this case the container instantiates the servlets in multiple JVMs. Although the servlet specification requires a sticky load-balancing strategy, container vendors may choose to implement instance-independent load balancing with distributed sessions and state. In such cases there's no guarantee that all requests from the same client session will be handled by the same JVM, and hence the same set of instances.
2. ***Swapping user sessions:*** Containers are free to swap user sessions and the associated load from one node to another in a cluster. In case of failure – say, a crash – of one JVM on one host, the cluster can also be configured to shift the load (and the session data) to another JVM (failover) or to restart the failed node.

Thus there's no guarantee that the same servlet instance will receive all the requests (from one user or all users) in a Web application.

Therefore, if not by specification, servlets are stateless by implication. What's the solution for handling state?

The best solution for handling request-specific state is to store it in the HttpSession object associated with the request. In the case of distributable servlet applications, the Servlet API 2.2 specification mandates that such state be serializable. It's good practice to ensure that all your session variables are serializable whether your application is distributable or not. Who knows? When your user base increases, you may want your Web application to be distributable.

For handling application-specific state (say, a list of addresses that's common to all users), servlet containers provide you with a ServletContext object. This object is similar to HttpSession in functionality. The main difference is that while an HttpSession object is specific to a client session, a ServletContext object is specific to a Web application on a given container. Irrespective of user sessions, all servlets in a Web application in a given container can access and share information via a ServletContext object.

What happens if your application is distributable and you want your application state stored in the ServletContext object to be shared across containers in a cluster? The servlet API doesn't provide for this.

The specification doesn't guarantee any kind of persistence of the state information (whether you store it in HttpSession or in ServletContext objects). In case you require persistent state management, it's better to devise your own mechanism, using some external data storage (files or databases), or delegate this to some other back-end component or system. Alternatively, check with your container vendor.

What about using static variables in servlets? There are three points to consider:

1. Are your static variables read-only? If not, you need to synchronize while updating such variables.
2. In case your static variables aren't read-only, you should also consider the effects of the distribution of the application. How do you make sure that static variables in different JVMs are consistent?
3. How do you protect your static variables/methods from other Web applications deployed on the same container (and JVM)?

To avoid these issues, consider storing such data in ServletContext objects.

SingleThreadModel — Why? And Why Not?

When and why should you implement the SingleThreadModel interface? This is a dilemma often faced by servlet developers.

Let me address the why part first.

Your servlets may implement the SingleThreadModel interface to make them (but not necessarily the resources that your servlets access) thread-safe. To quote from the API documentation: "If a servlet implements this interface, you are guaranteed that no two threads will execute concurrently in the servlet's service method. The servlet container can make this guarantee by synchronizing access to a single instance of the servlet, or by maintaining a pool of servlet instances and dispatching each new request to a free servlet."

Thus the specification guarantees that an instance of a SingleThreadModel servlet handles one incoming HTTP request at a time. Your servlets should implement this interface if you want to protect instance variables and other nonshareable resources from multiple request threads. But this doesn't guarantee that requests to a SingleThreadModel servlet will be handled sequentially. As discussed above, this is implementation-specific. Don't expect a "hit counter" servlet to work with a SingleThreadModel. This doesn't suffice.

The same effect can be achieved by synchronizing the various entry points (methods such as init(), service(), destroy(), etc.) into a servlet instance from the container. In this case the specification requires that containers serialize requests to that servlet instance instead of creating an

INSTANCES	THREADS	HOW TO ACHIEVE?
One	Multiple	Servlet that does not implement the SingleThreadModel interface
One	Single	Approach 1: Servlet that does not implement the SingleThreadModel with synchronized service() method Approach 2: SingleThreadModel servlet
Multiple	One thread per instance	SingleThreadModel servlet

TABLE 1 Instancing and threading model

instance pool, as is done in the case of SingleThreadModel servlets. However, you may not be required to take such an extreme step. For performance reasons it's better to narrow down the synchronization blocks in your code.

To summarize: three threading scenarios are possible (see Table 1).

What About Connection Pools?

How do you manage connections to databases and other resources/systems in a servlet-based Web application?

First, why do you need connection pooling? It saves the connection creation and disposal overheads. Once your application gets a connection to a relational database and finishes processing the database updates, it's better if it retains the connection object for future use. For efficient resource utilization you need connection pooling, which is a mechanism for recycling your connection objects.

Let's now examine some typical connection pooling strategies that servlet developers adopt. I draw these strategies from *Java Servlet Programming* and from various discussions from the Archives of the Servlet Interest Mailing List.

Connection Object as an Instance Variable

In this approach each servlet maintains its own connection object as an instance variable. The connection object is usually created in the init() method of the servlet. Thus each instance of the servlet holds a connection object. Look at the sample code in Listing 1. This approach is adequate for nontransactional database updates in the autocommit mode, since connection objects are specified to be thread-safe. But what happens if you want to implement multiple updates/inserts within a single transaction?

In the case of transactions your database server will associate each connection object with a single transaction. Consider the case of a servlet implementing a transaction. If an instance of this servlet is processing two concurrent requests in different threads, your transactions get mixed up. Instead of two transactions (as you'd expect), the database sees only one. Try the Java program in Listing 2. The TestThread class emulates a servlet instance processing two concurrent requests. In this class two threads are trying to do different transactions in two threads on the same connection object. You'll find that only those updates that occur after the rollback in the second thread will be committed to the database. Try it with different sleep intervals. The result? This approach doesn't preserve atomicity of transactions.

You can avoid this by synchronizing the transactional calls on the connection object as a group (try synchronizing the transact() method in Listing 2) or by implementing the SingleThreadModel interface, but your servlets will be penalized in terms of performance.

This solution isn't safe for transactional database access.

Connection Object Held in HttpSession

Instead of holding the connection object as an instance variable, you may want to store it in the HttpSession associated with a client so that all servlets serving a client may reuse the connection object. However, you should consider the following here:

- What happens if your servlet container chooses to serialize some of the session objects to conserve memory or to shift the load from one JVM to another in the same cluster? The serialization process would fail since connection objects aren't serializable.
- In case your application is distributable, the container requires that the session variables be serializable. If you try to pass a connection object to the HttpSession, the container may throw an IllegalArgumentException.
- What happens if your application is serving a thousand concurrent users? Your application would attempt to get a thousand connection objects from your database server.

As you can see, the foregoing solution isn't appropriate for developing production quality applications.

CONNECTION POOL MANAGERS

Another widely used solution for connection pooling is to develop a connection pool manager. Such a manager can take at least three forms: a singleton, a static class or a servlet with static attributes/methods. In all these cases the manager class would provide accessor methods to manage a pool of connection objects.

This solution may open up a major security breach because such a manager class/object is accessible from all servlets within the same JVM. If your servlet container is hosting multiple applications, there's nothing to prevent a rogue servlet from getting hold of one of the connection objects and destroying your database tables. Be wary of such a solution. You may need to provide an additional security mechanism (using the protection domains and principles of Java 2 security architecture) for returning connection objects only to trusted objects.

The DBConnectionBroker toolkit from Java Exchange recommends an alternative approach to the use of connection pool manager. This toolkit provides a DBConnectionBroker class to implement a connection manager and an HttpServletJXGB servlet that holds it as a static protected variable. This servlet is supposed to serve as a base class for all your servlets requiring database access. Instances of the derived servlet classes can therefore make use of the connection pool manager. In servlet containers before version 2.2, if servlets from multiple applications deployed on the same JVM extend from the same HttpServletJXGB class, there's still scope for the security issue mentioned above.

JDBC 2.0 OPTIONAL PACKAGE EXTENSION API

This Optional Package Extension API has several enhancements over the standard JDBC API. One of the enhancements is connection pooling. In this API the getConnection() method of the javax.sql.DataSource class returns connection objects. Depending on how the DataSource class is implemented, the getConnection method can return pooled connections. A significant implication of this approach is that the responsibility of connection pooling is delegated to the driver implementer. In addition to the above API, some of the JDBC driver vendors also implement connection pooling. Check the list of vendors at http://java.sun.com/products/jdbc/drivers.html.

Of all the approaches discussed in this section, database drivers complying with the JDBC 2.0 Optional Package Extension API offer the most robust solution for connection pooling.

Servlets and Operating Environment Resources

APPLICATION THREADS

Can a servlet start new threads? Not in all cases, as discussed below.

The environment in which servlet instances operate is guaranteed to be valid only during the course of a client request, that is, during the service() method. This environment includes the HttpSession, ServletContext, HttpServletRequest and HttpServletResponse objects. Make sure your application threads don't refer to these objects beyond the context of the service() method. In case you need specific information from these objects, consider copying such data into temporary objects (defined to suit your requirements) before starting any thread, and program the threads to use data from these temporary objects.

SECURITY

As indicated in the Servlet Specification 2.2, a servlet container may impose additional security restrictions on the accessing resources in the servlet environment. These resources include the JVM (threads, etc.), network, file system, and more. Although none of the container vendors seem to be moving in this direction currently, we can expect such restrictions from specialized servlet containers or domain-specific e-commerce products.

FILE SYSTEM

The servlet specification doesn't guarantee any specific "current working directory" from which you can access text files, configuration files, images, and so on. Two approaches can guarantee container-indepen-

dent file system access. The first approach is to keep all such resources accessible to the class path, and use the getResource() or getResourceAsStream() methods of the class loader to get the URL or an InputStream corresponding to resources. Alternatively, you can store the absolute locations of such resources on the file system as initialization (init-param) or context (context-param) entries in the associated deployment descriptor.

Summary

As Douglas Bennet points out, software is soft because of the soft programming constructs used to build software. There are many ways to implement a piece of functionality and many ways to use an API. Although most of these ways might work, not all of them may be resilient to changes. It's not always easy to grasp the implications of our designs, and there's always scope for debate. Nonetheless, there are better means to be considered that lead to better applications.

In this article I discussed some of the design practices with Java servlets. While most of the discussion is based on the Java Servlet Specification v2.2, by no means is it complete. The particular practices chosen for this discussion were motivated more by the need to probe into the intricacies involved in servlet development than on the practices per se.

References

1. Java Servlet Specification v2.2: http://java.sun.com/products/servlet/2.2/index.html
2. Archives of Servlet-Interest Mailing List: http://archives.java.sun.com/archives/servlet-interest.html
3. Hunter, J., and Crawford, W. (1998). *Java Servlet Programming.* O'Reilly & Associates.
4. Gamma, E., Helm,R., Johnson, R., and Vlissides, J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison Wesley.
5. Db Connection Broker: www.JavaExchange.com
6. The JDBC 2.0 Optional Package: http://java.sun.com/j2ee/bulletin/jdbc_2/extension.html
7. Bennet, D.W., (1996). *Hard Software: The Essential Tasks.* Manning Publications.

Author Bio

Dr. Subrahmanyan is a technical consultant with the electronic commerce division of Wipro Technologies, based in Bangalore, India. You can visit him at his Web site, www.Subrahmanyam.com.

subrahmanyam_avb@technologist.com

Listing 1: Connection Object as an Instance Variable

```
public MyServlet extends HttpServlet {
  private Connection connection;
  public init(ServletConfig config) {
    // Get database details from config
    ...
    // Create a connection object
    connection = ...
  }

  public void service(HttpServletRequest req, HttpServletResponse res) {
    // Use the connection object for database access
    ...
  }
}
```

Listing 2: Connection Object - One Transaction or Two Transactions?

```
import java.sql.*;
class Transaction {
  private Connection conn;
  Transaction() {
    String dburl = "jdbc:odbc:Test";
    String user = "scott";
    String password = "tiger";
    try {
       Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
       conn = DriverManager.getConnection(dburl, user, password);
       System.out.println("Connection created.");
    } catch(Exception e) { e.printStackTrace(); }
  }
  void transact(int action) {
    Statement stmt = conn.createStatement();
    try {
      conn.setAutoCommit(false);
      switch (action) {
      case 1 :
        stmt.executeUpdate("insert into dept values(41,'100','100')");
        System.out.println("1");
        try { Thread.sleep(5); catch(Exception e) {}
        stmt.executeUpdate("insert into dept values(51,'101','101')");
        System.out.println("11");
        stmt.executeUpdate("insert into dept values(62,'102','102')");
        System.out.println("111");
        try { Thread.sleep(2); } catch(Exception e) {}
        conn.commit();
        System.out.println("1 commited");
        break;
      case 2 :
        stmt = conn.createStatement();
        stmt.executeUpdate("update dept set dname='SomeAccounting' where deptno=10");
        System.out.println("2");
        stmt.executeUpdate("update dept set dname='SomeAccounting' where deptno=20");
        System.out.println("22");
        try { Thread.sleep(5); } catch(Exception e) {}
        conn.rollback();
        System.out.println("2 rolled back");
        stmt.executeUpdate("update dept set loc='Bangalore' where deptno=10");
        System.out.println("222");
        break;
      default :
        break;
      }
    } catch(Exception e) { e.printStackTrace(); }
  }
}

public class TestThread extends Thread {
  Transaction t;
  int action;
  public TestThread(Transaction tThread, int action) {
    t = tThread;
    this.action = action;
  }
  public void run() {
    t.transact(action);
  }
  public static void main(String s[]) {
    Transaction t = new Transaction();
    TestThread t1 = new TestThread(t, 1);
    TestThread t2 = new TestThread(t, 2);
    t1.start();
    t2.start();
  }
}
```

Enterprise JavaBeans: Architecture for the New Decade

WRITTEN BY TIMO SALO AND JUSTIN HILL

INTEREST + EFFORT + A WIDE RANGE OF AVAILABLE EJB BUSINESS COMPONENTS = AN ATTRACTIVE SERVER SOLUTION

Enterprise JavaBeans are being promoted as the component architecture for the new decade. The word *Enterprise* in the name would imply that EJBs are to the server environments what JavaBeans are to client computing. Both are component models, both are for Java, both try to deliver on the promise "write once, run anywhere." Beyond that, however, there is little commonality between them. EJB is not actually a software product but rather a specification of a server-side component architecture, to be implemented by vendors of server software. The specification combines object distribution with transaction processing and persistence. Its goal is to provide a standard architecture for building scalable, portable and distributed enterprise systems. The EJB architecture attempts to separate the business logic in the enterprise beans from the services that are required to host the beans. The promise of the architecture is that developers can focus on building the beans with little or no consideration given to the details of target runtime environments. According to the specification, the enterprise bean developer ships the bean with environment-neutral information on how to deploy it. Based on this information, the target environment expert adapts the bean to the environment using the environment-specific deployment tools.

The future potential of EJBs is great, hence the interest and effort being expended on them by many large server software vendors. The EJB architecture provides application software vendors with a single way to deploy the same application across a large number of very different server platforms, without their necessarily having to know all the details of writing applications on those platforms. Another side of the same coin is vendor-neutral standardization of server component software, which is important for enterprises with heterogeneous platforms. Even for organizations that aren't concerned about diverse environments, the EJB architecture and programming model can be used as a standard for server software development across different units.

This article provides a brief overview of the past, present and future of Enterprise JavaBeans. It also briefly describes an example EJB environment and its development tools. For space reasons we'll refrain from presenting any background information on transactions, security, persistence or various other distributed object concepts.

EJB Overview

There are two types of enterprise beans: *entity* beans and *session* beans. The former model business entities, such as a customer, an invoice or a contract. Entity beans are persistent, that is, they're stored in a database. Their persistence can be managed either by the runtime environment (EJB container) or by the bean itself. The advantage of the container-managed scheme is that the container handles all persistence and back-end peculiarities instead of having this knowledge embedded in application code. This scheme limits the beans' possible property types to a relative few, however, because the container accesses the properties directly when a bean is stored and restored. The only thing that can be assumed across various containers is that their persistence mechanisms support the basic Java types. With the bean-managed scheme the container doesn't access a bean's properties directly. The entity bean itself knows how to store and restore its state. Therefore the EJBs' internal composition structure can be whatever the developer desires and the container doesn't limit the property types. The downside of this approach is that the developer has to code all the database access. For a small, single application this is probably of little concern. For large applications this may lead to potential maintenance and performance problems.

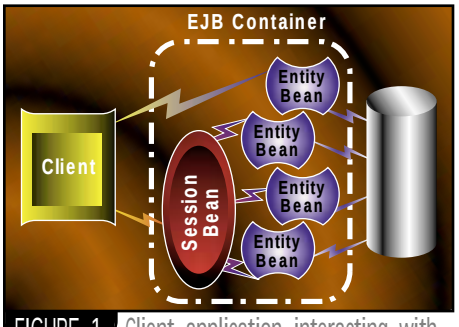


FIGURE 1 Client application interacting with session and entity beans

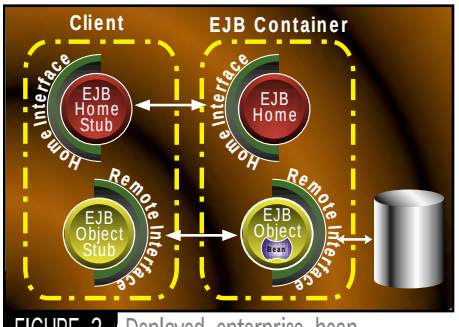


FIGURE 2 Deployed enterprise bean

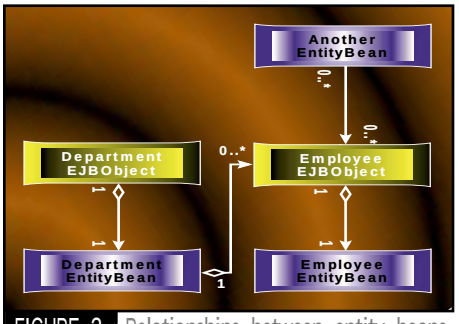


FIGURE 3 Relationships between entity beans and EJB objects

Session beans control workflow. They contain the logic for managing business processes and tasks, like "make a reservation" or "create a new customer." Due to their task-centric nature, the session beans are transient and not persistent. Session beans interact with entity beans and act as agents for client applications. A client application interacting with session and entity beans is presented in Figure 1.

Having the workflow logic in a session bean rather than in the client application reduces the network traffic and the number of connections needed. Session beans can be either stateful or stateless. *Stateful* session beans maintain their state across multiple client requests and are useful for long-running conversational tasks. Multiple clients, however, can't share the same bean instance because the bean's state depends on a particular conversation with a client. *Stateless* session beans don't maintain any state. Each method invocation is independent of any previous invocation and operates only on data passed in as parameters. The same bean instance can serve multiple clients, thereby reducing the amount of resources needed to serve the clients.

A complete enterprise bean package includes a description of how to deploy the bean in the target environment. This description includes the bean's interfaces, its persistent fields, how the bean's persistence is managed, its transactional behavior, and its security and environment properties. Based on the deployment description, the EJB container deployment tools create a wrapper (EJB object) and a factory (EJB home) for the bean that adapt the bean to the container. The deployment process also includes generation of all the distribution and persistence services for the bean. A deployed enterprise bean is presented in Figure 2.

An EJB container is a runtime environment for enterprise beans. The container's six primary services are naming, distribution, concurrency, transaction management, persistence and security. The container also provides resource management, including bean instance pooling and activation.

The EJB object implements the enterprise bean's remote interface. Clients can't access the bean directly; they can only access the EJB object that wraps the bean. The EJB object delegates the method invocations to the bean. This delegation scheme allows the container to intercept the messages between the client and the bean in order to manage transactions, security and persistence for the bean. Relationships between entity beans and EJB objects are presented in Figure 3.

An EJB home provides finder services for locating beans and factory/life-cycle services for creating and deleting beans. A home also provides metadata about the beans, including a bean's remote interface, its primary key class and its type (session or entity).

EJB Specifications

The EJB world, just like the rest of the Java world, is very specification-centric. The first EJB specification was released in the beginning of 1998. Version 1.0 defined the basic EJB components and the component contracts, including the container, homes, EJB objects, session and entity beans, and deployment descriptors. Version 1.0 also specified various other things, such as developer roles, exceptions, distribution, transactions, persistence and security.

The EJB model specified in version 1.0 certainly has many advantages. The first version, however, omitted many details and had a number of limitations that application developers and container providers had to work around. For example, version 1.0 leaves such essential OO concepts as inheritance and associations unspecified. This has caused either the tool vendors to add nonportable extensions to containers or the application developers to emulate these missing features in applications. Version 1.0 also states that entity beans are optional only, and due to the lack of an API for accessing the container's services the bean-managed persistence is nearly impossible to implement just by following the specification. It's also difficult to implement portable finders because the specification doesn't define any semantics for the finders.

The new 1.1 specification version doesn't introduce any major revisions to 1.0. It primarily introduces small enhancements here and there, which do, however, make a difference. Version 1.1 facilitates association implementations by specifying associated homes. The specification also opens doors for inheritance implementations by stating that enterprise bean classes can have superclasses. Some other key revisions are that entity beans are mandatory, finders can return collections in addition to enumerations, the security model has been refactored, deployment descriptors are stored in XML format rather than in binary format, and structural deployment information (e.g., home, fields) is separated from assembly information (e.g., security, transactional behavior).

An Example EJB Environment

IBM WebSphere Advanced Edition is an example of an enterprise server environment that hosts EJBs. The EJBs are developed and deployed using VisualAge for Java. The deployed EJBs are then executed within the WebSphere Application Server.

VisualAge for Java provides tools for EJB creation, packaging and deployment. It supports top-down, bottom-up and meet-in-the-middle approaches for developing EJB applications. With the top-down approach the database schema is created from existing EJBs. The bottom-up approach supports creating EJBs from an existing schema. The meet-in-the-middle approach is for situations in which existing EJBs are mapped to an existing schema. A screen capture of the EJB creation is presented in Figure 4.

The created EJBs are packaged into JAR files. VisualAge produces three types of JAR files: a generic portable JAR file, a predeployed JAR file for WebSphere Advanced Edition and an EJB client JAR file. The deployment step includes mapping EJBs to a database schema, generating the CRUD services for container-managed persistence, generating the EJB objects, generating EJB homes and corresponding finders, and generating the distribution services (i.e., stubs and skeletons).

VisualAge for Java also includes EJB client and server environments for testing and debugging EJBs. The EJB server is a lightweight version of the WebSphere Application Server that runs within the VisualAge IDE. VisualAge can automatically generate a test client for a deployed EJB and run it against the test server. This significantly simplifies and speeds up testing of EJBs.

WebSphere Application Server Advanced Edition is a highly scalable EJB server that supports multiple platforms, databases and transaction systems. Its workload manager provides server clustering and application-level workload management and distribution across mul-

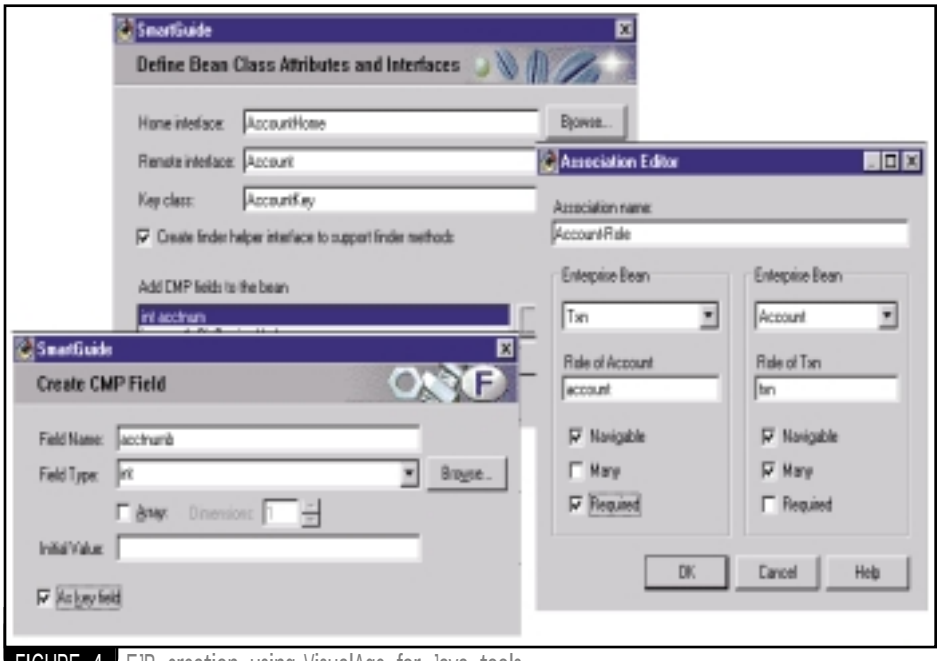


FIGURE 4 EJB creation using VisualAge for Java tools

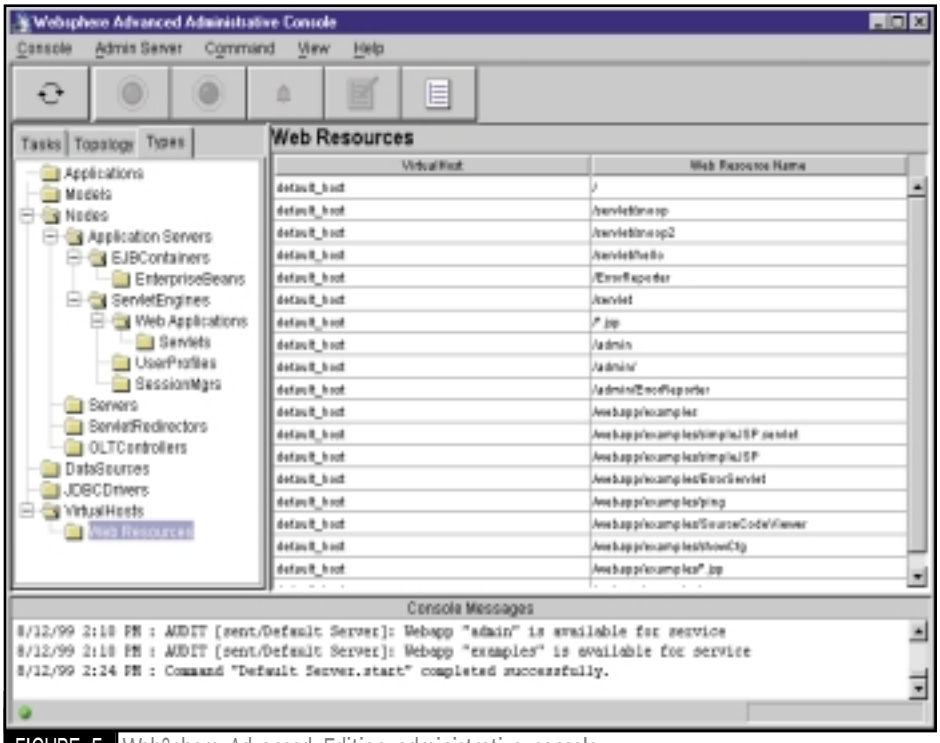


FIGURE 5 WebSphere Advanced Edition administrative console

tiple servers. The distributed transaction management executes across multiple applications and components. The distributed security operates at EJB level and can utilize third-party authentication and secure association services. The user registries are LDAP-based and the access control lists and policies can be established at the user and group levels, and for specific calls or methods within applications. Advanced Edition includes remote administration, logging and analysis capabilities, and it also can be monitored with Tivoli-based tools. A screen capture of the Advanced Edition administrative console is presented in Figure 5.

The server also provides a wide range of capabilities for interacting with enterprise databases, transaction processing systems and other applications.

The application server supports both container- and bean-managed persistence. The messaging between EJBs is optimized for both remote and local situations: if both the sender and the receiver EJB reside in the same container, the messaging bypasses the Object Request Broker. This optimization is especially important with applications that consist of large numbers of fine-grained objects.

The Future of EJBs

The first two EJB specification versions were written primarily by Sun. Subsequent development, however, has been done within the Java Community Process. Two EJB-related specifications are currently being developed: the EJB 2.0 specification and a specification for UML/EJB mapping. Expert groups for both began their work at the end of last year and first drafts can be expected later this year.

The EJB 2.0 will be a major revision to the specification. The Java Specification Request delineates several important objectives for the specification. The most important ones are full support for inheritance and associations, and portable query syntax for finder methods. The other objectives include integration with the Java Message Service, improved entity bean persistence, additional methods on the home interface and a mechanism for extending containers. The JMS integration will allow asynchronous method invocation from clients, which is important for systems that have part-time disconnected clients. The persistence enhancements include data access components that encapsulate the database access, and a standard API that enables persistence tools to operate across different containers. The additional methods in the homes are for providing behavior that is independent of finding and managing individual bean instances. For example, a home might have a batch-update method that operates on all instances within the home. The container extension mechanism, which is based on an interceptor pattern, provides means for specializing the container's behavior for specific operational environments, thus reducing the need to have several versions of the base container itself.

The objective of the UML/EJB Mapping Specification is to provide means for modeling EJB applications using UML. The specification defines a set of standard UML extensions expressed as a UML profile. This profile describes the relationship of EJB constructs and a deployment descriptor to UML model elements, the forward engineering transformation from the model elements to EJB implementation artifacts, and the reverse engineering transformation from the implementation artifacts to the model elements. The specification also defines a mechanism for associating UML models and EJB implementation artifacts that are stored in the same JAR file. The mechanism is based on XML DTD and can be used for reflection and tool automation.

Conclusion

Even though Enterprise JavaBeans provide an advanced server architecture, they won't make server application development trivial. Building high-performance enterprise servers, with or without EJBs, is never simple. Still, the EJB specification represents a significant step forward in the systematic development of scalable and distributed component-based applications. The advantages of unifying on a single server architecture and programming model for Java applications are enormous, even for applications that aren't distributed in nature or deployed on different server platforms.

The specification is by no means perfect or complete as yet. Where the specification is lacking, developers and vendors must make up the difference or do without. This can mean guessing the direction the specification is evolving, developing additional functionality and, later, creating migration strategies from the proprietary enhancements to some future version of the specification.

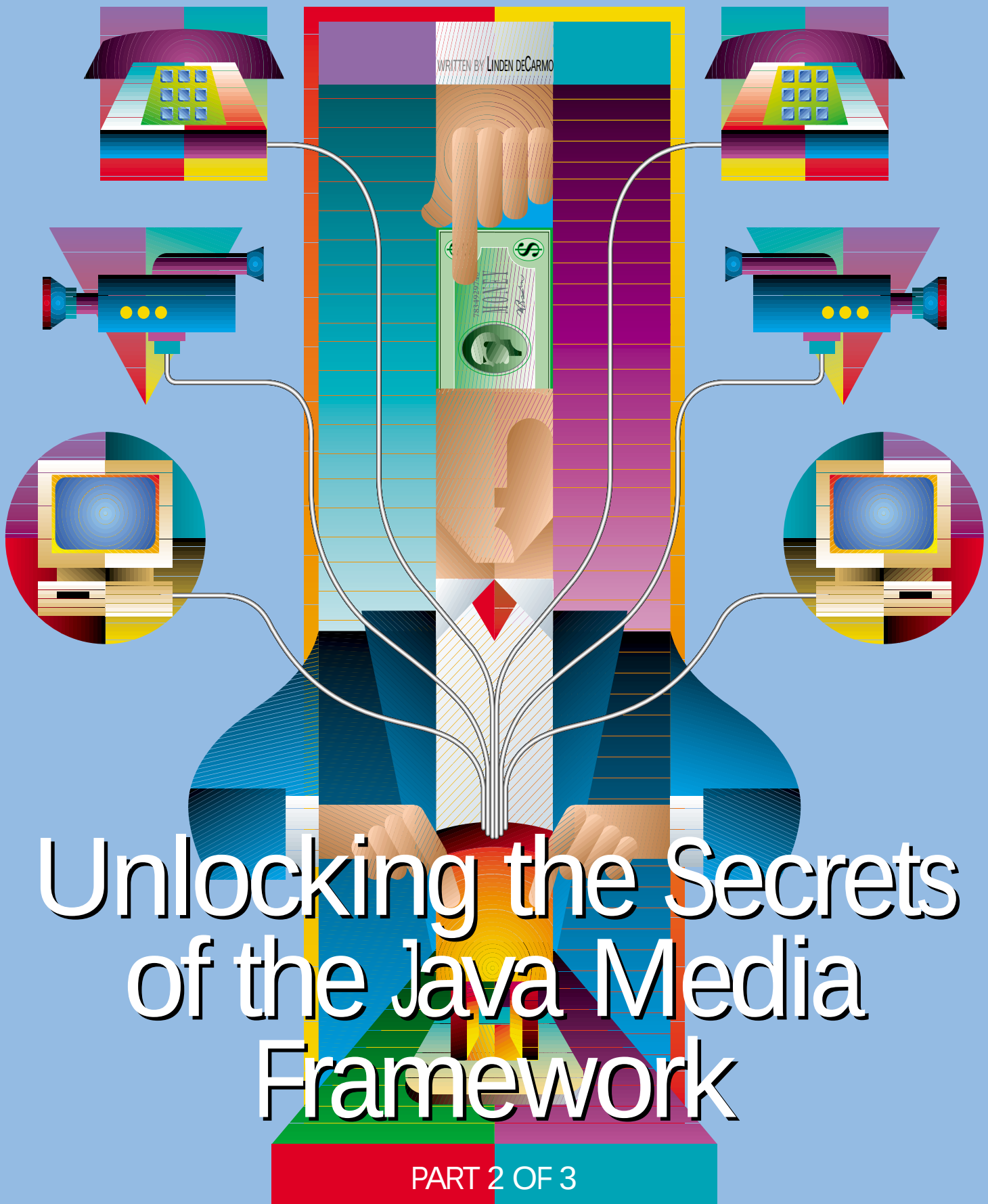
It's unlikely that there will be a viable alternative to EJB for a server component architecture in the near future. Most major server software providers offer EJB environments, and there's a lot of general interest and effort being invested in EJBs throughout the software industry. This, added to the fact that there's already a wide range of EJB business components available from independent software vendors, makes EJBs one of the most attractive server solutions today.

Author Bios

Timo Salo, a senior software engineer for IBM Transarc, has been developing large-scale, multitier distributed systems for banking and financial industries since 1985. Timo is currently architecting EJB persistence tools for various IBM development environments, including VisualAge for Java and WebSphere Application Server.

Justin Hill, a senior software engineer at IBM, is currently working on Smalltalk and Java feature development. Since 1987 he has been developing object-oriented systems for the banking, telecommunications, process manufacturing and insurance industries. Justin has specialized in object persistence and object-oriented query languages for the last five years.

tjsalo@us.ibm.com / jhhill@us.ibm.com



WRITTEN BY LINDEN DECARMO

Unlocking the Secrets of the Java Media Framework

PART 2 OF 3

RTP and RTSP: Protocols that address the transportation of multimedia content over IP

The Internet is strewn with multimedia minefields. Lost or out-of-sequence packets and transmission delays can create havoc in your applications. Fortunately, you can overcome these problems by using protocols optimized for multimedia transportation. This article explains why these protocols are necessary, and examines how the JMF implements them and how you can use them to spice up your programs.

Transportation Woes

The TCP/IP (Transmission Control Protocol/Internet Protocol) dominates IP (Internet Protocol) traffic because of its reliability. This reliability is possible because TCP/IP contains numerous software layers that prevent packet loss and ensure that packets arrive in the order they were sent.

Unfortunately, while TCP/IP's reliability is beneficial for textual data, it can have a devastating impact on multimedia streaming. This occurs because its numerous software layers steal critical processing power away from multimedia devices. Furthermore, reliability is unnecessary for most multimedia applications and may force disturbing pauses in the playback.

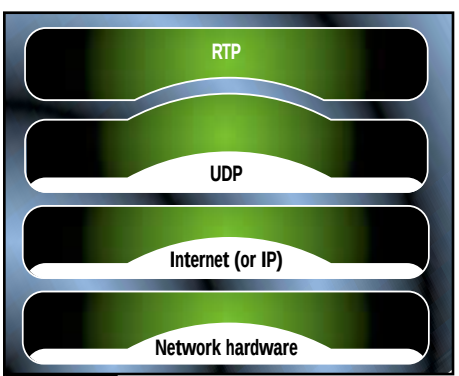


FIGURE 1 The RTP protocol uses UDP to transmit real-time multimedia content.

By contrast, the UDP (User Datagram Protocol) is a lightweight but unreliable protocol that's ideal for multimedia transport. Because it doesn't guarantee that packets will arrive at their destination or that they will arrive in sequence, UDP consumes less processing power than TCP/IP. Although unreliability sounds scary, few UDP packets are ever lost. Should a packet be lost, multimedia CODECs (Compressors/Decompressors), the software algorithms that process multimedia data, gracefully compensate to ensure smooth playback.

Although you can use UDP to transmit audiovisual content, it isn't optimized for multimedia use. Consequently, the RTP (Real-Time Protocol) is built on top of UDP and it provides time stamps, synchronization, packet-loss detection and other multimedia features (see Figure 1). Since RTP is a binary protocol, you must examine packet headers to determine multimedia attributes such as audio or video CODEC or sampling rate.

RTP may be deployed as a stand-alone protocol or as part of a higher-level protocol. For example, the RTSP (Real Time Streaming Protocol) uses RTP to transmit audio and video packets. However, you control RTSP with text-based commands such as PLAY, STOP and PAUSE (see Table 1). When the RTSP subsystem receives one of these commands, it updates its internal state and, if necessary, changes the type of RTP packets it is transmitting.

It's All in the Packaging

Last month we examined how you can play popular file formats such as QuickTime or .WAV with JMF. Unfortunately, these programs struggle with Internet content because they require that the entire file be downloaded before commencing playback. Not only does this ruin interactivity, but the gargantuan file sizes make this approach impractical. Therefore, streaming media content is specifically optimized to enable immediate playback and is not bound by file-size restrictions.

Since there's no file format, RTP streams can be identified by individual packet headers. For instance, RTP packets contain media type and frame size headers. The media type header indicates the audio or video CODEC that's being transmitted (see Table 2). RTP supports numerous audio CODECs, but most audio packets contain PCM (Pulse Code Modulation) or a PCM variant such as Mu-law or A-law. Similarly, you can stream a variety of video CODECs with RTP, but most JMF applications use H.261.

The frame size header represents logical subdivisions of data, and each packet contains

a whole number of frames. Larger frame sizes take longer to capture and transmit, and therefore may cause audio gaps as your application is waiting for data. By contrast, extremely small frame sizes can overwhelm your application with tiny packets and can be difficult for audio CODECs to decode. Consequently, if you're sending RTP packets, you should choose a frame size that balances interactivity and processing power.

Take Control of the Situation

RTP streams are grouped into entities called sessions. A session contains two or more devices exchanging multimedia content. Since RTP is built on an unreliable protocol, there's no method to determine whether a session's RTP packets are being transported successfully. Thus RTP is usually combined with RTCP (Real Time Control Protocol), which lets you monitor network traffic and track the session's participants.

The Real Thing

Now that we've discovered how multimedia content is transported over IP, it's time to examine the JMF streaming APIs. Since the RTSP programming model is similar to the JMF devices we discussed last month, we'll examine it first.

RTSP COMMAND	ACTION
PLAY	Starts media playback.
PAUSE	Pauses media playback.
SETUP	Initializes an RTSP session.
TEARDOWN	Closes an RTSP session.

TABLE 1 RTSP uses text commands to manipulate multimedia streams

ENCODING NAME	AUDIO/VIDEO DATA
PCM	Audio
1016	Audio
G726-32	Audio
GSM	Audio
G723	Audio
DVI4	Audio
DVI4	Audio
LPC	Audio
PCMA	Audio
G722	Audio
L16	Audio
L16	Audio
QCELP	Audio
MPA	Audio
G728	Audio
DVI4	Audio
DVI4	Audio
G729	Audio
CN	Audio
CelB	Video
JPEG	Video
nv	Video
H261	Video
MPV	Video
MP2T	Audio/Video
H263	Video
dynamic	Application Specific

TABLE 2 List of audio and video formats that can be transported with RTP

RealNetworks uses RTSP as core protocol for its RealPlayer and is a strong advocate for RTSP in various Internet task forces. In an effort to make RTSP and RealMedia content (e.g., .ra, .ram) more pervasive, they've provided a JMF-based RTSP runtime and SDK.

The first thing you'll notice about RealNetworks RTSP support is how smoothly it fits into the JMF architecture. In fact, the applet we created last month can play RealMedia content without modification. Although it's possible to use vanilla JMF APIs with RTSP, the RealNetworks Player provides exciting enhancements that you'll want to exploit in your applications.

To access this new functionality, you ask the Manager to create a Player that uses RealMedia content. Then you detect the presence of extra features by seeing if the Manager returned a Player, which is an instance of com.real.media.RMPlayer. The following example illustrates how you can detect the presence of an enhanced RealNetwork Player. If it's a com.real.media.RMPlayer, you have access to pause functionality and automated media position information.

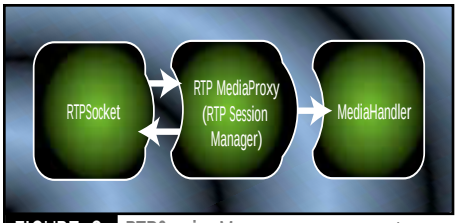


FIGURE 2 RTPSessionManager components

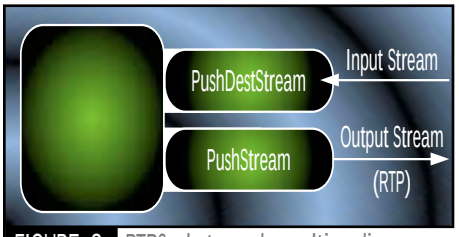


FIGURE 3 RTPSocket sends multimedia content to client MediaHandlers and streams out session and RTCP information.

```
// create a Player to preview the selected
file

player = Manager.createPlayer(mrl);

// check to see if this is RealAudio con-
tent....
if ( player instanceof
com.real.media.RMPlayer)
{
    // if it is, we can take advantage of
this
    // functionality....
    rmPlayer = ( com.real.media.RMPlayer )
player;
}
```

JMF Players and Controllers do not surface a pause() method because they were designed to play content on a local machine. When you call the Player's stop() method, it flushes (or removes) buffers from its associated device and enters stopped state. When playback is restarted, these buffers can be refilled rapidly and streamed to the device.

By contrast, it can take several seconds to prefetch a Player that is streaming remote content. Thus you can't flush these buffers when a Player is stopped and still have a responsive device. For these scenarios the RMPlayer surfaces the pausePlay() method.

pausePlay() pauses playback, but it doesn't flush buffers from the RMPlayer's audio/visual device (see the code below). This ensures that playback can resume instantly when the user hits the play button. An additional benefit of this approach is that playback can restart exactly where it was originally paused. Resumption after a stop() is less exact because the Player can detect only the approximate location where it stopped.

```
// pause rather than stop -- no data loss
rmPlayer.pausePlay();
```

Like other Controller methods, pausePlay() is asynchronous. When the RMPlayer finishes pausing, it notifies your listener method with a RMPauseEvent (see code below). Never assume that the pause has completed until this event is received.

```
else if (event instanceof RMPauseEvent)
{
    // insert code to handle RMPauseEvent
here
}
```

Besides pause capabilities, the RMPlayer provides automated stream position information via the RMOnPosChangedEvent. The RMPlayer sends an RMOnPosChangedEvent every 100 milliseconds and you can use the event's getPositionInNanos() method to uncover the active media time (see code below). This approach is superior to polling since you don't waste processing cycles trying to guess the current media time.

```
else if (event instanceof RMOnPosChanged-
Event)
{
    // insert code to handle RMOn-
PosChangedEvent here
}
```

Diametrically Opposed

Unlike RealNetworks RTSP solution, Sun has designed a hybrid RTP architecture. Part of the solution is integrated with JMF, while the other portion resides outside JMF. The core element of this hybrid approach is the RTPSessionManager, which supports traditional JMF objects such as DataSources and MediaHandlers. However, it adds additional RTP-specific features such as performance metrics and media type detection.

The RTPSessionManager is a supervisory object that manages all RTP sessions you wish to control with JMF. It enables playback of RTP content via the self-contained DataSource and MediaHandler (see Figure 2).

The DataSource provided by the RTPSessionManager is called RTPSocket. Like a conventional DataSource, RTPSocket streams packetized RTP content to client MediaHandlers. However, RTPSocket has one unique attribute: it can also output data. For example, you can use it to transmit RTCP data or session statistics (see Figure 3).

When you request that the JMF Manager construct a Player to handle RTP content, it searches for a MediaHandler that is compatible with the RTPSocket. Normally, the only such MediaHandler is the RTPSessionManager. This occurs because the RTPSessionManager is a special type of MediaHandler called a *MediaProxy*.

A MediaProxy manipulates content and forwards the modified content to a subsequent MediaHandler. Thus they're usually pure soft-

ware entities, not associated with a particular hardware device. The portion of the RTPSessionManager that implements the MediaProxy interface translates (or depacketizes) RTP packets into flat data buffers that other Players or MediaHandlers can process. The JMF Manager then searches for a MediaHandler that can accept the output of the depacketizer. If a compatible MediaHandler can be found, the stream can be played.

MediaLocators to the Rescue

The process of constructing an RTP-enabled Player is dramatically more complex than creating a simple audio or video player. Fortunately, the Manager hides this complexity by letting you construct a RTPSocket with a MediaLocator (see code below). The MediaLocator must be in the following format:
rtp://IPaddress:port/mediatype where the *IPaddress:port* combination is the address of the RTP session, and *mediatype* represents the type of content in the session (i.e., audio or video).

```
if ((mrl = new MediaLocator(rtpServer)) == null)
{
    System.err.println("Can't build RTP MRL for " + rtpServer);
}
```

After the Manager gives you a reference to the RTPSocket object, you should pass the reference to the Manager's createPlayer() method, as follows:

```
try
{
    // Create a Player with the rtp DataSource....
    rtpPlayer = Manager.createPlayer(rtpsource);

}
```


Because RTP sessions are sensitive to network traffic, it's crucial that you monitor session status. RTPSocket provides access to this information by exposing an RTPControl object. If you call RTPSocket's getControl() method with the appropriate string, it will return a RTPControl object as follows:

```
 // we'll use this name to query the rtp player for an
// rtp control.
private static final String rtpControlName = "javax.media.rtp.RTP-
Control";
RTPControl rtpcontrol;

// get the RTP control from the DataSource
rtpcontrol = (RTPControl) rtpsource.getControl(rtpControlName);
if (rtpcontrol == null)
{
    System.out.println("No RTPControl interface.");
}
```

RTPControl is a cornucopia of information. There are methods to retrieve the number of packets lost or received out of sequence (see code below). Furthermore, you can query global information such as the total number of bytes processed or the number of invalid packets detected.

```
// get the statistics info from the rtp control object
RTPReceptionStats stats = rtpcontrol.getReceptionStats();
String packetslost = "\tPackets lost: " + stats.getPDUlost() +
"\n";
String outoforder = "\tPackets out of order: " + stats.getPDU-
MisOrd() + "\n";
String packetsreceived = "\tPackets received: " + stats.get-
PDUProcessed() + "\n";
```

Peeking Under the Hood

If you want minute control over how your RTP session is created, you can't rely on the Manager to create the RTPSessionManager for you. Rather, you must perform all the steps yourself. This includes allocating a new RTPSessionManager, attaching yourself as a listener to the RTPSessionManager and initializing the session parameters. Then you must start the session, create a DataSource for it and construct a Player for the DataSource. Unless you're a control freak, or absolutely must control low-level session attributes such as RTP description lists, it's easier to let the JMF Manager handle these details for you.

Until Next Time

RTP and RTSP are protocols that specifically address the transportation of multimedia content over IP. RealNetworks RTSP Player gives you access to RealMedia content and requires no modifications to your JMF programs. RealNetworks also provides custom pause and position change information necessary for streaming applications.

Unlike RealNetworks' pure JMF solution, Sun's RTP architecture is a JMF hybrid. It's possible to write RTP applications using only JMF. However, if you want detailed control over the RTP session, you'll need to manipulate the RTPSessionManager outside the realm of JMF.

Next month we'll unveil the truth about JMF 2.0 – the most anticipated Java multimedia product ever released.

AUTHOR BIO

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

lindend@mindspring.com

Debugging and Optimization Techniques

WRITTEN BY ALAN WILLIAMSON

I'd like to introduce you to a new ***JDJ*** series consisting of selected excerpts from my current book, *Java Servlets: By Example*. I've put a number of chapters into article format, hoping they'll give you some insight into the world of servlets and the sort of things we have to do at the server side. The first one really has nothing to do with servlets at all – it's more of a general overview of debugging and optimization techniques. This is by no means an exhaustive exploration, merely a toe-tipper.

Next month we'll begin our discovery into the world of servlets.

• • •

Java is a programming language. It is a programming language that runs programs known as *class files*. A class file is a sequence of instructions that perform some logical task. Sometimes these instructions run as expected...and sometimes they don't.

With most IDEs (integrated development environments) some form of debugger is available for use in developing applications or applets. However, due to the very nature of servlets, setting up a debugging environment isn't always easy.

This article presents a general debugging class that will aid you as the developer to quickly locate and fix problems. Just because Java makes coding easy doesn't mean that all standard coding practices go out the window. This article looks at a number of simple steps that can be done to speed up your class objects.

Debugging

Debugging programs is one of those annoying things that the majority of us have to perform at some point in our development life. As soon as we write a piece of code, be it a class or a method, we like to think it is perfect. "That won't need testing," we say to ourselves. However, nine out of 10 times it does require testing and sometimes quite intensive detective work. Many people dislike this process of debugging, fixing problems. It's even worse if you have to try and fix somebody else's code. Many developers view this as a necessary evil that simply has to be done. But they loathe it.

It's a state of mind. Think of it as a big game…the thrill of the chase. Somewhere in there, lurking under lines and lines of code, a small bug is causing the whole system to come crashing down. It becomes a game to try and flush him (or her – no reason why a bug can't be female!) out into the open. When thought of with this mind-set, that horribly small bug you've been avoiding suddenly becomes so much more attractive.

Many techniques for debugging are available to developers. Most rely on the tools provided by their development environment. In general, they allow the inspection of variables, stepping into code, freezing output and modification, to name but a few of the features available. Some rely on the old-fashioned method of debugging: print statements.

The print statement method is coming back into vogue. It has never had it so good. Thanks to Java, developers have the ability to print information that used to be privileged information of the debugger and compiler.

The most common types of bugs are the ones that are the most obvious when found. For example, a wrong variable name or incorrect assignment can leave us kicking ourselves for using it in the first place. The debugger generally tells us which line the program crashed in by highlighting it when it crashes. Very handy.

With the combination of Exception and Throwable, Java has given the developer the power to do this themselves. By honing in on the line that's causing the problem, these tools can build up a better picture of the bug as a whole.

Java uses the System.out.println(...) method to display messages on the console. Every object can be printed as a string, and this makes life so much easier.

However, the console isn't much use to servlets. Sure, at development time the developer can monitor the console, but more times than not the bug is highlighted through heavy use or after a period of time, when the console isn't being monitored.The servlet API defines a class method that allows the developer to send text to the main log file used by the Web server. This gives the developer a certain level of control, but still doesn't give that instant access to information that may be critical in the catching of a bug.

The class presented here gives all the functionality required to fully utilize the print statement method. This class has:

- *Easy access throughout the virtual machine*
- *File logging*
- *Socket logging*
- *Exception logging*

The main advantage of this class, as you'll discover, is that it's not restricted for use in the servlet environment. Any Java environment can use the features of the following debugging class.

Debugging Class

One of the most important things about any debugging class is simplicity. It can't impact the overall performance of the program. A class that takes up too much processing time can't be used as a debugging tool. This is the developer's equivalent of the age-old problem facing engineers who build measurement tools: how to build a tool that will accurately measure a given flow, substance, mixture, temperature – whatever – without interfering with the original environment in which the measurement will be taken. A system that measures the rate of water flow will unintentionally slow the flow down slightly by its very presence. A rod that measures temperature will either heat or cool the particles or atoms around about it. The smaller the impact, the better the tool.

Debugging is no different. If we wish to find a bug during development, this isn't considered a major problem. The tool is allowed to impact the environment if necessary. However, if it's to ship with a production version, to catch any of those long-term bugs it has to have minimum impact.

The class presented here serves both those criteria.

Class Structure

Let's assume that the class will be used throughout the virtual machine. Let us further assume that we don't want to keep a specific object instant to the class – we simply want to use it. We can achieve this by making the methods static, which means we don't ever have to instantiate the class manually.

Since we know that objects provide a toString(...) method, which returns a string representation of the object, we'll only support the ability to print strings. The calling method can then control exactly what it wishes to print. We can outline the class as shown below:

```
public final class Debug extends Object {
  private Debug(){}

  public static void println( String _line
){
    //-- do some processing
  }
}
```

Notice the constructor. We've declared it private to safeguard against anyone creating an instance of it. This ensures that only one instance of the class will exist in the virtual machine. Couple that with the fact that we've made our class final means that no one can extend it, that is, use it as a base class for another.

Simple Console Output

One of the most primitive things we want to control is whether or not the output is sent to the console. We can control this through a simple Boolean value, bSystemOut, that when set true will display all output on the console. We can write our core output function with this ability, as shown in Listing 1.

Notice the provision of an extra variable, bOn. This gives developers control of all debugging processes through the use of one variable. They can turn off all debugging with a single method call.

Irrespective of the output method, the same string will be displayed. This will be the original string from the developer with the current date inserted at the beginning. Having the date included in the output is very handy when it comes to debugging threads, for example.

Listing 2 shows the methods available to the developer for controlling the output of the debugging class.

File Output

Having output sent to the console is useful, but not always practical. For example, it would require watching the console for any messages, as they're lost once scrolled by. This of course assumes you don't redirect the console to a file, but this is generally operating system-specific.

Sending the output to a file is a much more convenient way for the developer to trace what has been printed out. We can build this functionality in to our debugging class very easily.

One of the things we have to ask of the file feature is not to overwrite any existing debug information. This is a fairly important feature, as subsequent runs of the servlet or application may wipe out valuable debug information.

Many of the standard file-handling classes that come as part of the Java libraries don't handle writing to the end of a file. The RandomFileAccess class, however, allows us to open a file and start appending data to it.

To make the class more efficient, we'll open the file once and leave it open for subsequent writes. This requires us to hold a reference to it throughout the life cycle of the debugging class. By initializing this variable with the value of null, we can easily determine whether the file is already available. If it isn't, we'll open it and move the file pointer to the end of the file so all output will be appended. The following listing illustrates this.

```
if ( bFile ){
  try{
    if ( OutFile == null ){
      OutFile = new RandomAccessFile(
filename, "rw" );
      OutFile.seek( OutFile.length() );
      OutFile.writeBytes( "\r\n------Log-
ging Restarted------- n-ary limited v1.3 --
-------\r\n" );
    }
    OutFile.writeBytes( D );
    }catch(Exception E){}
  }
}
```

Notice again how we control the use of the file, through a variable called bFile. This particular piece of code will be placed inside the core print routine after the call to the system out.

This class assumes it will open the same file each time. Therefore the name of the file can be fixed in the class, and for this case all file output will go to a file "debug.txt". This file will be created in the current directory of the running application.

Socket Output

So far we have the ability to send output to the console and to a file. Wouldn't it be nice if we could log on to a port from anywhere and view the output as and when it happens?

Java has made the answer to this sort of question very trivial. Adding such networking capabilities to a class is no big task. We want to give the developer the ability to connect to a known port and then to view all the output through a standard TELNET session.

We need two things to handle this: (1) a thread that will listen for incoming client connections, and (2) a class that will handle the communications for each client.

Let's define the method for setting up a listening socket. We want to handle many client connections at once, so we'll create a simple loop that will listen for a connection and, once connected, create a class to handle that connection and then return to listen for more connections.

Since we already have a class – our debugging one – there's no point in creating another. So we'll extend this class to use the Thread class, and define a run() method that will be used to listen for client connections. Listing 3 shows this.

For us to send out all messages to all our clients we need to keep a reference to each client. This is achieved through the Vector class, which keeps a reference to the clientDebug class. This is the class we're going to use to handle each client, as shown in Listing 4.

The first thing this class does when it's created is to attempt to create an instance of DataOutputStream, which gives us a means to easily send strings to the socket. Since this is purely an output class, there's no need to get an input stream to the socket. If this fails, an exception will be thrown and the class reference will be set to null.

If it's successful, then some information regarding the current operating system and available memory is printed. This serves as both an informative read and also to confirm to the client that a good connection has indeed been made.

The debugging class will print to socket by calling the println(...) method. If the client is no longer available or has disconnected, an exception will be thrown. This will cause the method to return a false result, and the debugging class knows to remove the client from the list of available clients.

The method shown in the following code details the output to the client connections. It sets up an Enumeration to the Vector, which will allow it to run through the list easily. If the method returns false, then it's removed from the list.

```
private static void printToClients( String
D ){
  Enumeration E = clients.elements();
  clientDebug CD;
  while (E.hasMoreElements()){
    CD = (clientDebug)E.nextElement();
    if ( CD.println( D ) == false )
      clients.removeElement( CD );
  }
}
```

The following code controls the socket connections. At the core print routine, if the Vector that holds the client's connection is null, it's assumed the server hasn't been started. The server is then started by creating an instance of the debugging class and calling the start() method. This will invoke the thread and call the run() method.

```
if ( bSocket ){
  if ( clients == null ){
    new Debug().start();
    clients = new Vector();
  }

  if ( clients.size() > 0 )
    printToClients( D );
}
```

After it's been created, it calls the method shown in the previous snippet.

Exception Handling

In general, the time you need a debugging class is when things start going wrong. This usually manifests itself in lots of exceptions being thrown. Having the ability to handle these exceptions properly would increase the usefulness of a debugging class.

One of the really nice features of Java is the ability to display a complete stack trace. This is where each called method leading up to the

problem is displayed, complete with the line number, if available. This information alone can save many hours of debugging time.

When an exception is thrown, the stack trace is available. This is through a method call from the printStackTrace() method. The output from this method is sent to the standard error stream. Therefore we must redirect it in order to get a copy of this invaluable data.

The next listing illustrates this redirection. Instead of sending the output straight to an output stream, we'll store it in a string and send it on through our normal print routine. This way the developer will still control the flow of output.

```
public static void printStackTrace( Excep-
tion E ){
  ByteArrayOutputStream OS = new ByteAr-
rayOutputStream();
  PrintStream ps = new PrintStream( OS );
  System.setErr( ps );
  E.printStackTrace();
  System.setErr( System.err );
  println( OS.toString() );
}
```

Sending a stream to a string is a trivial matter. This involves creating a new instance of a ByteArrayOutputStream, which will hold all the data. Once collected, we simply call the toString() method and get the contents of this buffer returned as a string.

Using the Class

Now that we have the class built, let's look at how we can use it. The following code will most definitely throw an exception since the variable Temp is null.

```
try{
  String Temp = null;
  Temp = Temp.toLowerCase();
catch(Exception E){
  Debug.println( "This will throw an
exception" );
  Debug.println( E );
  Debug.printStackTrace( E );
}
```

When this occurs, a number of methods of the Debug class are called. In the default state this would mirror all the print statements to the console, file and any existing client connections.

Complete Source

Listing 5, the complete code for the debugging class, can be found on ***JDJ***'s web site, www.JavaDevelopersJournal.com.

Optimizing for Size

We can split this section into two further subsections – common sense, and not so obvious. First of all, a commonsense suggestion for reducing the size of a class includes naming variables and methods with smaller names.

Java has given us the ability to use really long method names, and while this is handy, they have to be stored in the class file. Reducing this can significantly decrease the size of the class file.

The Java library is a rich tapestry of classes that perform many tasks, and with over 1,500 classes to chose from, there's a lot of scope. So don't rewrite any functionality that may already exist.

Seems an obvious one, but you'd be surprised at the number of developers who have redeveloped standard classes that were unknown to them at the time of their rewriting. This reduces your code complexity, but in addition, the chances are good that the class you're using may be a native class and therefore run much faster.

Reuse methods. If you've developed a lot of different methods that don't really need an object instance to operate, place them in a class of their own and declare them as static methods. This will reduce the need to repeat them in classes that need them.

Now for some hints on the not-so-obvious tricks. One of the most common things you'll see in code is where strings are added together using the "+" operator. This is convenient, but also very slow. The compiler will generally replace each one with an instance of StringBuffer(...). For each "+" operator a new instance of StringBuffer will be created. Replace the operators with one instance and use the append(...) method for adding strings together.

Storing dates can be a pain, especially if you're moving between databases. Instead of storing the date, store the millisecond equivalent in a long. Not only does this save space in the database table and the virtual machine, but it makes querying on the dates very efficient as you're simply comparing two numbers as opposed to two dates.

Finally, remember to compile with optimizations turned on. This will automatically try to reduce your code size by eliminating dead code and converting some methods into inline calls.

Optimizing for Speed

Optimizing for speed follows the same principle. But always remember to do it – before and after timings. This way you can confidently convince yourself that your work has actually made an improvement and not by some fluke increased the execution time…which has been known to happen.

One of the most expensive operations you can do is exception handling. Try to replace exception blocks with logical tests, if possible, and reduce the number of lines contained within a block.

The synchronized method used for ensuring thread-safe execution is also a large overhead. Minimize the use of this where possible.

Creating objects in Java is a breeze. But they cost. Try to reuse objects as opposed to creating new ones. For every new one you create, the memory has to be allocated the garbage collector can come along and clean up all unused objects. This is common in loops.

If you're doing a lot of dividing, think of reworking your logic so you can divide by 2. Dividing is a very expensive operation, and if you can divide by 2, this can be replaced with shift operations. Many game developers optimize their code to take this into account and ignore the remainder that may be lost.

Warning

Before you modify all your code, remember that it's important to get it working first. Optimizing as you develop is never a good idea as this more often than not distorts the logic, which makes it harder to find unwanted features, or bugs. Many of the optimizations above will make a difference, but use them wisely. There's no need to completely redo every class you wrote just to save a couple of bytes. Use them wisely.

Author Bio

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

alan@sys-con.com

Listing 1: Modification of the Core Output Routine

```
static boolean bOn = true;
static boolean bSystemOut = true;
static SimpleDateFormat DateFormat = null;

public synchronized static void println( String Line ){
 if ( !bOn )
  return;

    if ( DateFormat == null )
    DateFormat = new SimpleDateFormat( "dd/MMM HH:mm.ss: ");

  String D = DateFormat.format(new java.util.Date()) + Line +
"\r\n";

  if ( bSystemOut )
    System.out.println( D );
}
```

Listing 2: Controlling the Output

```
public static void On(){
  bOn=true;
}
```

N-ary

www.n-ary.com

```
public static void Off(){
  bOn=false;
}

public static void SystemOn(){
  bSystemOut=true;
}

public static void SystemOff(){
  bSystemOut=false;
}
```

Listing 3: Listening for client Connections

```
public final class Debug extends Thread {
  static Vector clients = null;
  static int SOCKET_PORT = 2000;

  public void run(){
    Socket sIN;
    ServerSocket sSERVER;

    try{
      sSERVER = new ServerSocket( SOCKET_PORT );
    }catch(Exception E){
      return;
    }

    for(;;){
      try{
        sIN = sSERVER.accept();
        clients.addElement( new clientDebug( sIN ) );

    }catch(Exception E){}
  }
}
}
```

Listing 4: Handling the Client Connection

```
class clientDebug {
  private Socket sIn;
  private DataOutputStream out;

  public clientDebug(Socket _sIn){
    sIn = _sIn;
    try{
      out = new DataOutputStream( sIn.getOutputStream() );
      out.writeBytes( "------Logging Restarted------- n-ary
limited v1.3 ---------\r\n" );

      out.writeBytes( "[os.name]      = [" + System.getProp-
erty("os.name") + "]\r\n" );
      out.writeBytes( "[os.arch]      = [" + System.getProp-
erty("os.arch") + "]\r\n" );
      out.writeBytes( "[os.version]   = [" + System.getProp-
erty("os.version") + "]\r\n" );


      out.writeBytes( "[java.version] = [" + System.getProp-
erty("java.version") + "]\r\n" );
      out.writeBytes( "[java.vendor]  = [" + System.getProp-
erty("java.vendor") + "]\r\n" );

      Runtime RT = Runtime.getRuntime();
      out.writeBytes( "[total memory] = [" + RT.totalMemory()
+ " bytes]\r\n" );
      out.writeBytes( "[free memory]  = [" + RT.freeMemory()
+ " bytes]\r\n" );
      out.writeBytes( "----------------\r\n" );

    }catch(Exception E){
      out = null;
    }
  }

  public boolean println( String _D ){

    try{
      out.writeBytes( _D );
      return true;
    }catch(Exception E){
      return false;
    }
  }
}
```

Python Programming in the JVM —*Continued from page 66*

- ***String parsing:*** In addition to the string parsing that we've shown in this article, Python has libraries for regular expression string parsing, slice-notation syntax and other great features that make string parsing easy. Score 10 of 10
- ***Productivity:*** Python has an extensive class library that makes doing common tasks easy. In addition, Python has built-in language support for collection objects including collection literals that let you define a collection. These language constructs and class libraries make programming strikingly productive. Score 10 of 10
- ***Working well with Java classes and APIs:*** In JPython you can instantiate Java classes, invoke Java methods, subclass Java classes and interfaces, and easily set up bean events. In addition, JPython has support to work with JavaBean properties and you can compile JPython into Java classes. Thus you can create JavaBeans, servlets and applets in JPython. Score 10 of 10
- ***Development environment/debugging:*** It's good to have an interactive interpreter, and JPython has a good one. However, if you're used to having GUI builders, debugging in an IDE, setting watches, and so forth, forget about it. The development environment for JPython is its Achilles' heel. Unlike Python, which has some mature IDEs, JPython has nothing. If JPython had an IDE like JBuilder or Visual Basic, Java would have a serious competitor for the most popular language for the JVM. Hey, Java IDE makers – Borland, Symantec, NetBeans (or should I say Corel, BEA, Sun) – get busy. (*Rumor mill:* There's one major IDE maker already considering supporting JPython.) Score 2 of 10

Parting Shots

Python, a high-level, dynamic, object-oriented language, is the easiest language to learn – easier than Visual Basic. JPython, which is very close to Python, has been certified 100% Pure Java.

JPython has a lot of momentum, and its syntax is mean and lean. It's to JavaBeans what Visual Basic is to ActiveX, and interest in it is growing. A recent poll at the NetBean's Web site showed JPython as the leading Java scripting language. In our poll at ***JDJ***, JPython is neck and neck with NetRexx.

Components have a symbiotic relationship with high-level languages. For example, Visual Basic did well because of ActiveX components. And ActiveX did well because of tools like Visual Basic. On the Java platform we have the component models; we need the glue, that is, tools for the high-level languages – debuggers, IDEs, and the like. JPython makes good glue, and it transcends the role of a glue language.

For more information on JPython see Guido Van Rossum's article at www.developer.com/journal/techfocus/081798_jpython.html. See also www.jpython.org/ and www.python.org/.

Many of the examples in this article were based on examples in my book, which is scheduled for publication in April from Addison Wesley Longman.

Author Bio

Rick Hightower currently works at Buzzeo Corporation (www.buzzeo.com), the maker of ZEOLogix, an EJB application server, rules engine and workflow. He is a principal software engineer working on an EJB container implementation and distributed event management. In addition, he is author of a book, Programming the Java APIs with JPython, *to be published by Addison Wesley.*

rick_m_hightower@hotmail.com

ADVERTISING INDEX

ADVERTISER	URL	PH	PG
4TH PASS	WWW.4THPASS.COM	877.484.7277	37
AMERICAN CYBERNETICS	WWW.MULTIEDIT.COM	800.899.0110	35
APPLIED REASONING	WWW.APPLIEDREASONING.COM	800.260.2772	71
CAREER CENTRAL	WWW.CAREERCENTRAL.COM/JAVA	888.946.3822	86
CAREER OPPORTUNITY ADVERTISERS		800.846.7591	102-113
CONCENTRIC NETWORK	WWW.CONCENTRICHOST.NET	800.476.0196	89
DEVELOPMENTOR	WWW.DEVELOP.COM	800.699.1932	97
ELIXIR TECHNOLOGY	WWW.ELIXIRTECH.COM/DOWNLOAD/	65 532.4300	41
EMBARCADERO	WWW.EMBARCADERO.COM/ADMINISTER		85
EMBARCADERO	WWW.EMBARCADERO.COM/DESIGN		87
EMBARCADERO	WWW.EMBARCADERO.COM/DEVELOP		83
EVERGREEN INTERNET, INC.	WWW.EVERGREEN.COM		43
FIORANO SOFTWARE, INC.	WWW.FIORANO.COM	408.354.3210	45
FLASHLINE	WWW.FLASHLINE.COM	216.861.4000	39
GEEK CRUISES	WWW.GEEKCRUISES.COM	650.327.3692	97
GENERIC LOGIC, INC.	WWW.GENLOGIC.COM	413.253.7491	10
HOTDISPATCH.COM	WWW.HOTDISPATCH.COM		4
IAM CONSULTING	WWW.IAMX.COM	212.580.2700	61
IBM	WWW.IBM.COM/DEVELOPERWORKS		67
IIDEA INTEGRATION	WWW.IDEA.COM	703.821.8809	29
INETSOFT TECHNOLOGY CORP	WWW.INETSOFTCORP.COM	732.235.0137	73
JAVACON2000	WWW.JAVACON2000.COM		78-79
JDJ STORE	WWW.JDJSTORE.COM	888.303.JAVA	100-101
KL GROUP INC.	WWW.KLGROUP.COM/REAL	888.328.9596	77
METAMATA, INC.	WWW.METAMATA.COM	510.796.0915	55
MICROSOFT	MSDN.MICROSOFT.COM/SUBSCRIPTIONS		11
MICROSOFT	MSDN.MICROSOFT.COM/WINDOWSDND		13
N-ARY	WWW.N-ARY.COM		95
NEW ATLANTA	WWW.NEWATLANTA.COM	678.366.3211	53
NUMEGA	WWW.COMPUWARE.COM/NUMEGA	800.4-NUMEGA	31
OBJECT DESIGN	WWW.OBJECTDESIGN.COM/JAVLIN	800.962.9620	58-59
OBJECTSWITCH CORPORATION	WWW.OBJECTSWITCH.COM/IDC35/	415.925.3460	51
OPTIMIZE IT	WWW.OPTIMIZEIT.COM		33
PERSISTENCE	WWW.PERSISTENCE.COM		17
POINTBASE	WWW.POINTBASE.COM/JDJ	877.238.8798	25
PRAMATI	WWW.PRAMATI.COM/J2EE.HTM	914.876.3007	69
PROTOVIEW	WWW.PROTOVIEW.COM	800.231.8588	3
QUICKSTREAM SOFTWARE	WWW.QUICKSTREAM.COM	888.769.9898	49
RIVERTON SOFTWARE CORPORATION	WWW.RIVERTON.COM	781.229.0070	93
SEGUE SOFTWARE	WWW.SEGUE.COM/ADS/CORBA	800.287.1329	20-21
SIC CORPORATION	WWW.ACCESS21.CO.KR	822.227.398801	75
SLANGSOFT	WWW.SLANGSOFT.COM/CODE/SPIRUS.HTML		19
SOFTWARE AG	WWW.SOFTWAREAG.COM/BOLERO	925.472.4900	47
SYBASE INC.	WWW.SYBASE.COM	800.8.SYBASE	15
SYS-CON	WWW.SYS-CON.COM	800.513.7111	34
TIDESTONE TECHNOLOGIES	WWW.TIDESTONE.COM	800.884.8665	27
TOGETHERSOFT LLC	WWW.TOGETHERSOFT.COM	919.772.9350	6
VISICOMP, INC.	WWW.VISICOMP.COM	831.335.1820	57
VISUALIZE INC.	WWW.VISUALIZEINC.COM	602.861.0999	88
VSI	WWW.BREEZEXML.COM	800.556.VSI	65
YOUCENTRIC	WWW.YOUCENTRIC.COM/NOBRAINER	888.462.6703	63

JDJ NEWS

Gemstone Expands Product Family

(*Beaverton, OR*) – GemStone Systems, Inc., announces an expanded portfolio of its Java-based GemStone/J application server products.

The new GemStone/J Web Edition, Component Edition, Enterprise Edition and Commerce Automation Edition are fully code-compatible, a first in the application server industry. They use varying levels of the same J2EE environment including JSPs, servlets and EJBs, along with GemStone's unique Java virtual machine pooling and shared memory environment to support peak performance of iCommerce applications. www.gemstone.com

Red Hat Brings Java Technology to Linux Community

(*Research Triangle Park, NC*) – Red Hat Inc. and IBM announce a worldwide licensing and distribution agreement for IBM's Java software for Linux.

Under the agreement Red Hat will license and distribute IBM's Java Runtime engine, Java Virtual Machine and the IBM Developer Kit for Linux, Java Technology Edition. IBM's Java Technology will be distributed with the Red Hat Linux Operating Systems Enterprise Edition. Red Hat will provide worldwide support contact for users of the IBM Java Technology as they create and deploy Java-based Internet solutions on Red Hat Linux. www.redhat.com

TurboLinux Licenses IBM's Java Technology for Linux

(*San Francisco, CA*) – TurboLinux announces that it has licensed IBM's Developer Kit for Linux, Java Technology Edition, that includes a Java Virtual Machine giving customers the ability to enhance database performance, simplify application installation and minimize the resources needed to maintain their databases. TurboLinux will include IBM's Developer Kit for Linux in the next release of its products. www.turbolinux.com

Inprise Launches Visibroker for Java 4.0

(*Paris, France*) – Inprise/Borland launches VisiBroker for Java 4.0, the Linux version of its CORBA ORB. Based on open industry standards, VisiBroker 4.0 delivers the ideal foundation for customers to expand their presence on the Web and provides them with the technology infrastructure needed to support enterprise-scale e-business applications. A free 60-day evaluation version can be downloaded from the Inprise/Borland Web site. www.borland.com/visibroker

HiT Software Ships First DB2 Access for Java 2 Platforms

(*San Jose, CA*) – HiT Software, Inc., releases HiT JDBC/DB2 v2.0, a high performance Java JDBC level 2 connectivity middleware for DB2 server access. JDBC level 2 compliance offers Java developers powerful new data access techniques including scrollable result sets, batch updates, programmatic updates, character stream support for international Unicode and time zone support.

HiT JDBC/DB2 now supports SSL v3.0 for authentication and data encryption of data across networks. www.hit.com

MediaFORM Introduces SmartDRIVE2

(*Exton, PA*) – MediaFORM introduces SmartDRIVE2, the first 12X compatible CD-R drive made for professional duplication systems. Key features include Copy Protection; Watermarking; SmartSTAMP; SmartMEDIA; SmartRID; Frame Accurate Recording; and Mini-Disc and Business Card CD-R Compatibility. www.mediaform.com

The Open Group Announces TETware 3.4

(*Menlo Park, CA*) – The Open Group introduces TETware 3.4, the latest release of the Test Environment Toolkit, a multiplatform test framework.

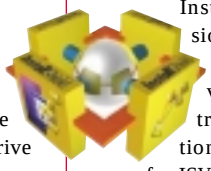
TETware 3.4 provides a Java testing environment that permits the development of Java applications with test classes. Once implemented, these classes can be automatically exercised by the test case manager, communicating with TETware's test case controller using the standard mechanisms. www.opengroup.org

Flashline.com Announces QA Testing Service

(*Cleveland, OH*) – Flashline.com announces the formation of the Flashline Quality Assurance Lab, an independent service to test third-party JavaBeans components, Enterprise JavaBeans and Java code for structure, performance, server-side capacity and other quality issues. Flashline has formed agreements with industry-leading companies to use their best-of-breed testing tools to provide objective and precise code analysis. This service offers IT professionals quick and thorough measurement procedures, and enables higher quality, more robust and efficient components by testing Java code throughout the development cycle from the initial design process through deployment. www.flashline.com

InstallShield Ships Professional 2000 Second Edition

(*Schaumburg, IL*) – InstallShield Software Corporation releases InstallShield Professional 2000 Second Edition, the latest version of its industry-leading installation-authoring solution for ISVs and corporate developers. www.installshield.com



Persistence to Port PowerTier for EJB to Linux

(*San Mateo, CA*) – Persistence Software is porting PowerTier for Enterprise JavaBeans to the Linux operating system, offering a powerful solution for e-commerce sites.

PowerTier for EJB provides stateful failover, partitioning and workload balancing through PowerTier's ability to synchronize multiple object caches distributed across multiple servers – features that will greatly extend the capabilities of a Linux-based environment. Red Hat Linux 6.1 – the latest release of the Red Hat Linux operating system – incorporates easy installation, software update information,and access and improved system management capabilities. www.redhat.com www.persistence.com

Keenovation Introduces eSign-on

(*Alexandria, VA*) – Keenovation, Inc., introduces the Internet's first Java-based Web companion aimed at automating logins and registration, and filling out forms for online purchases.

With a single mouse click eSign-on users can securely and quickly log in to multiple Web-based e-mail accounts, check the status of their online orders at various online shopping sites or simply access their online electronic and full-service brokerage accounts. A free copy can be downloaded from the company's Web site. www.keenovation.com