JDJEDGE CONFERENCE & EXPO FINAL PROGRAM INSIDE P.91

JAVA™ DEVELOPER'S JOURNAL

The World's Leading Java Resource

Volume:7 Issue:6

JAVADEVELOPERSJOURNAL.COM

THE LARGEST WEB SERVICES CONFERENCE & EXPO IN THE WORLD!
web services EDGE
JDJEDGE
XML EDGE
www.sys-con.com
Conference: June 24-27 Expo: June 25-27
FINAL PROGRAM
.NET
The Future of Enterprise Computing
Call Today!
JACOB JAVITS

FINAL CONFERENCE PROGRAM
▸ *INSIDE* PAGE 91

RETAILERS PLEASE DISPLAY
UNTIL AUGUST 31, 2002

$5.99US $6.99CAN

0 09281 01751 6 07>

SYS-CON MEDIA

Sonic Software

www.sonicsoftware.com

Zero G

www.zerog.com

Apple
Computer, Inc.

www.apple.com/macosx

Apple
Computer, Inc.

www.apple.com/macosx

BEA

www.bea.com/download

JAVA DEVELOPER'S JOURNAL

INTERNATIONAL ADVISORY BOARD

• CALVIN AUSTIN *(Lead Software Engineer, J2SE Linux Project, Sun Microsystems)*, • JAMES DUNCAN DAVIDSON *(JavaServlet API/XMP API, Sun Microsystems)*, • JASON HUNTER *(Senior Technologist, CollabNet)*, • JON S. STEVENS *(Apache Software Foundation)*, • RICK ROSS *(President, JavaLobby)*, • BILL ROTH *(Group Product Manager, Sun Microsystems)*, • BILL WILLETT *(CEO, Programmer's Paradise)* • BLAIR WYMAN *(Chief Software Architect IBM Rochester)*

EDITORIAL

EDITOR-IN-CHIEF: ALAN WILLIAMSON
EDITORIAL DIRECTOR: JEREMY GEELAN
EXECUTIVE EDITOR: NANCY VALENTINE
J2EE EDITOR: AJIT SAGAR
J2ME EDITOR: JASON R. BRIGGS
J2SE EDITOR: KEITH BROWN
PRODUCT REVIEW EDITOR: JIM MILBERY
FOUNDING EDITOR: SEAN RHODY

PRODUCTION

VICE PRESIDENT, PRODUCTION AND DESIGN: JIM MORGAN
ASSOCIATE ART DIRECTOR: LOUIS F. CUFFARI
EDITOR: M'LOU PINKHAM
MANAGING EDITOR: CHERYL VAN SISE
ASSOCIATE EDITORS: JAMIE MATUSOW
GAIL SCHULTZ
JEAN CASSIDY
ASSISTANT EDITOR: JENNIFER STILLEY
ONLINE EDITOR: LIN GOETZ
TECHNICAL EDITOR: BAHADIR KARUV, PH.D.

WRITERS IN THIS ISSUE

BILL BALOGLU, MICHAEL BARLOTTA, JASON R. BRIGGS, KEITH BROWN, ANDREW CHALK, FRED DAOUD, SAMUDRA GUPTA, JEFF HEATON, SAMI LABABIDI, BILLY PALMIERI, BILL RAY, DAVID ROSENSTRAUCH, AJIT SAGAR, ALAN WILLIAMSON, BLAIR WYMAN, JOE XU

SUBSCRIPTIONS:

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

SUBSCRIPTION HOTLINE: SUBSCRIBE@SYS-CON.COM

COVER PRICE: $5.99/ISSUE
DOMESTIC: $49.99/YR. (12 ISSUES)
CANADA/MEXICO: $79.99/YR. OVERSEAS: $99.99/YR.
(U.S. BANKS OR MONEY ORDERS). BACK ISSUES: $10/EA., INTERNATIONAL $15/EA.

EDITORIAL OFFICES:

SYS-CON MEDIA 135 CHESTNUT RIDGE RD., MONTVALE, NJ 07645
TELEPHONE: 201 802-3000 FAX: 201 782-9600

JAVA DEVELOPER'S JOURNAL (ISSN#1087-6944) is published monthly (12 times a year) for $49.99 by SYS-CON Publications, Inc., 135 Chestnut Ridge Road, Montvale, NJ 07645. Periodicals postage rates are paid at Montvale, NJ 07645 and additional mailing offices. POSTMASTER: Send address changes to: JAVA DEVELOPER'S JOURNAL, SYS-CON Publications, Inc., 135 Chestnut Ridge Road, Montvale, NJ 07645.

©COPYRIGHT:

Copyright © 2002 by SYS-CON Publications, Inc. All rights reserved. No part of this publication may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopy or any information storage and retrieval system, without written permission. For promotional reprints, contact reprint coordinator Carrie Gebert, carrieg@sys-con.com. SYS-CON Publications, Inc., reserves the right to revise, republish and authorize its readers to use the articles submitted for publication.

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

ALAN WILLIAMSON EDITOR-IN-CHIEF

To Our CFML Friends: We Welcome You

"Fly me to the moon...let me walk among the stars" or at least America. I am at present sitting in a Continental plane flying over the beautiful Scottish islands, sun beaming in through the window, contemplating the week ahead of me. My destination is Toronto where I will be attending the CFML conference, CFNORTH.

It has been very interesting to watch how the CFML community is reacting to the Java releases. Their responses serve to illustrate the point I have been banging on about for the last few months: outside of the Java community, we still have a long way to go in the education and perception of our beautiful language. Allow me to give you another example of ignorance. As luck would have it, the May issue of the in-flight magazine has an article on Web technologies. This article is, of course, aimed at the masses, but let me give you its paragraph on what JSP is:

Like ASP, a specification that allows a browser to ask the database for information. For example, people play games against each other on the Internet on JSP pages. Advantages: works well behind the scenes. Disadvantages: whereas ASP pages are programmed with PERL, JSP pages are programmed with a very specific set of tools – one for which it may be harder to find expert maintenance.
—Continental, May 2002

Read it? Do me a favor, read it again. What do you think? Not really selling JSP is it? In fact, you would probably get a bad feeling about JSP. As we know, the author is wrong; if anything JSP is the easiest way to get your hands on Java – no compiling, just save and surf! How difficult is that I ask you?

Keep your eyes open. Look outside your Java circle and ask those who haven't the in-depth knowledge you have what their perception of Java is. I think you'll be surprised by the responses.

The CFML language has enjoyed years of success as a server-side application implemented in C/C++. Therefore, the move to the Java platform is a controversial one, raising an eyebrow or two within CFML's loyal user base. The posts by respected CFML people regarding the move to Java are startling; I urge you to check out the official CFML mailing list archives for examples of posts that will send your blood racing.

We have a barrage of posts from people berating Java, claiming it could never be as fast as a native implementation, and it's no wonder CFMX is running slow. I'm assuming these people haven't given CFMX a chance. In our tests, CFMX outperforms the CF5 version running on the same hardware. So where these people are getting their facts from is beyond me.

What has happened is that the old Chinese whispers network has begun. The focus of conversation has moved from CFML to whether or not Java is up to the task. We all know that Java is more than up for the job of serving dynamic content; this was its calling and this is where Java is making serious headway, at the server side.

Although it appears as if we, the Java community, are the only people who really know this for sure. Our message doesn't seem to be getting out there.

There's a huge movement within the CFML community to come to Java and sit at our table to share in the delights we offer. This is wonderful and we welcome and encourage our new developers.

More power to you.

AUTHOR BIO

Alan Williamson is editor-in-chief of Java Developer's Journal*. During the day he holds the post of chief technical officer at n-ary (consulting) Ltd, one of the first companies in the UK to specialize in Java at the server side. Rumor has it he welcomes all suggestions and comments.*

alan@sys-con.com

TogetherSoft Corporation

www.togethersoft.com/challenge/1

An Interview with Jason Hunter

Open sourcing Java – a Q&A session

JDJ caught up with Jason Hunter, Apache Software Foundation vice president, after JavaOne to discuss the major announcement regarding the controversial issue of open sourcing Java. Jason holds a seat on the JCP Executive Committee overseeing the Java platform.

JDJ asked readers to pose their questions to Jason, so you can hear it straight from the horse's mouth!

<alan>: What does this announcement mean?
<jason>: It heralds an agreement between Apache and Sun regarding open source implementations of Java standards. Among the items agreed upon:

- For Sun-led specifications finalized from here forward (including revisions to existing specifications) and for key specifications already released, the license terms will allow independent implementations under open source licenses.
- The Test Compatibility Kit (TCK) binaries for these specifications will be made available at no cost to qualified open source, nonprofit, and academic groups.
- A three-member board, including a representative from the Apache Software Foundation and a representative from academia, will ensure an impartial qualification process.
- Sun will provide substantial support to aid these qualified groups in the use and execution of the TCKs.
- Specification leads will be allowed to choose open source licenses for their Reference Implementations (RIs) and TCKs if they desire.

There's an additional item in the agreement that will be of special importance to commercial implementers of Java specifications:

- Sun will make the TCKs for future and key past specifications available separately from the RI code. Previously, there had been a requirement to license the RI when licensing a TCK, which made true independent implementations impossible.

For details on the agreement, see Sun's Letter of Intent at http://jcp.org/aboutJava/communityprocess/announce/LetterofIntent.html and Apache's press release at http://jakarta.apache.org/site/jspa-agreement.html.

Is Java finally open sourced?
[anonymous]
<jason>: That depends on what this question means. I believe it can mean three different things.

One: "Can there be an open source Java implementation?" The answer is yes, this agreement should ensure that when J2SE 1.5 is released it can have an independent, open source licensed implementation. Many Java specifications will have that ability right away.

Two: "Is the reference implementation for Java (the JDK) available as open source?" This agreement does not address that. Apache doesn't believe in telling Sun or anyone else what license their code must be under. What Apache believes is that if you don't like a code license, you must have the ability to code it yourself. The purpose of this agreement is to allow that.

Three: "Will Java be developed following the open source method?" Right now there's a formal process by which Java specifications are created within the Java Community Process (JCP). The JCP will probably remain as a formal specification-creating body, although Apache hopes it will become more open. The project I lead, JDOM, is JSR number 102, and with the JSR we're proving you can have open development on a Java specification even within the JCP. Now, the clear ability to have open source RIs and TCKs will help this JSR progress.

Apache saw an explosive growth in the number of projects during the last couple of years. What do you see as the major challenges facing the Apache Foundation in being able to maintain the momentum and support all of the existing efforts in the long term.
[Kirk Pepperdine, kcpeppe@sprintmail.com]
<jason>: Apache only accepts projects that have an established, vital, and active community. Put another way, code must come with a community of coders. Apache doesn't accept "throw it over the wall" code – no matter how enticing.

So far the policy has worked well. The main challenge is to effectively integrate the coders into the Apache community.

Are you completely satisfied with the substance and level of commitment contained in this announcement?
[kcpeppe@sprintmail.com]
<jason>: Right now it's an agreement on a course of action. Apache will be much more satisfied when the plan of action has been executed.

In your opinion, what does Sun stand to gain from this agreement?
[kcpeppe@sprintmail.com]
<jason>: Apache has been a strong partner of Sun, promoting Java on the server side. Projects like Tomcat, Ant, Xerces, Xalan, and Axis/SOAP are widely used and vital to the server-side Java community. Many Apache projects have Sun developer support, quite a few have implemented Java specifications, and usually when that happens the Apache code has been used as a base for the specification RIs. However, the legal issues surrounding open source Java implementations absolutely had to be settled for Apache to continue supporting the Java Community Process (JCP) and the specifications the JCP produces.

Does Apache plan on creating a JXTA implementation?
[wwsmithms, wwsmithms@yahoo.com]

<jason>: JXTA is not a JCP specification. It's a Sun-backed but independent and language-neutral open source project with an Apache-style license. It's unaffected by this agreement.

Do you think that this agreement will result in Sun finally testing JBoss for J2EE certification?
[eric]
<jason>: JBoss will now have access to the TCKs for J2EE 1.4 coming later this year. Because JBoss is a for-profit corporation, they will probably have to pay for the TCK. When talking with them they had no objections to that. It'll be up to them to ensure compliance with the TCK.

How does Apache feel/cope when a lot of their ideas/products are incorporated by other companies? For example, an equivalent of the ORO project has been incorporated into Java 1.4 as regexp classes.
[Fintan Conway, Fintan.Conway@mail.esb.ie]
<jason>: There are two ways to interpret this question.
One: "How does Apache feel when its products are incorporated into products of other companies, e.g., how IBM WebSphere uses the Apache Web Server?" Apache uses this particular license (a very liberal license that allows reuse in commercial products) precisely because Apache wants its code to be both useful and used by anyone and everyone who is interested.
Two: "How does Apache feel when the JCP creates specifications for the Java platform that are in a similar vein to preexisting Apache projects?" This question is more difficult to answer because there may be as many opinions as developers within Apache. I think it's fair to say this: people are glad to see Java incorporating functionality such as regular expressions and logging. However, we've seen the specifications being created in private and with little community input, compared to the typical, Apache project.
Not surprisingly, when you compare the Java specifications with the Apache projects that were created "in the wild" with large amounts of user feedback, the new standards may not seem of equal quality. What's more, the incorporated technology will only fall farther behind as Apache continues innovating and doesn't have to wait until J2SE 1.5 for the next release. At the same time, Apache people realize there's the Microsoft Windows–style effect: whatever's provided with the platform gains the majority marketshare, regardless of its comparative quality.

Is this the beginning of the end? Are we entering a future where we'll be seeing different flavors of Java, like we've seen in the C/C++ world, and building specific versions for specific target JVMs?
[George]
<jason>: It's important to recognize that the agreement provides for "compatible" open source independent implementations. Apache believes this agreement will improve Java's portability through the wider availability of Test Compatibility Kits and with the existence of solid open source implementations on which vendors can build. Servlet portability has only improved with the availability of the Tomcat servlet engine.

It may seem like an obvious question but even though Apache is a nonprofit organization, how do they plan to continue to deliver excellent, quality offerings in the future?
[Mark Menzies, Mark.Menzies@avenida.co.uk]
<jason>: For a good description of how open source projects work, I would recommend a book like The Cathedral and the Bazaar [by Eric S. Raymond and Bob Youngor], or a research paper like "Open-Source Software Development and Distributed Innovation" at www.smu.edu.sg/research/pdfs/kogut06K1.pdf.

Can you explain the finer print on the support that is on offer from Sun for the open source community?
[Mark.Menzies@avenida.co.uk]
<jason>: In the months before the agreement Apache set as a minimum requirement that the TCK binaries be "easily available" to any open source or academic group. The notion that a specification lead could decide to waive TCK license fees was not sufficient, because a busy spec lead might not even respond to a small group or individual, and we wanted to allow for the next Linus Torvalds to spring up (this time in Javaland). During negotiations, Sun was willing to license the TCKs free of charge to these groups, but was concerned that many TCKs are so complicated they essentially require support in order to run. Sun couldn't grant support to all nonprofits because, unlike TCK binaries, there's a real cost for each support customer.
To solve that problem, the proposal is to have a board (consisting of a Sun member, an Apache member, and an academic member) who would grant free TCK binaries to any qualifying group, and grant TCK support within some budget limits. The method for qualifying groups and granting support will be worked out by the board members. I personally wouldn't be surprised if open source mailing lists became the common vehicle for support for open source groups.

How do you think the major Java players feel about this move?
[Paul]
<jason>: Among the JCP Executive Committee (EC) members there has been strong support for this change. Apache's concerns were listed by EC members as the number one issue to be dealt with in the JSPA legal document as it goes from community review toward public review. Several EC members went on record in support during JavaOne.

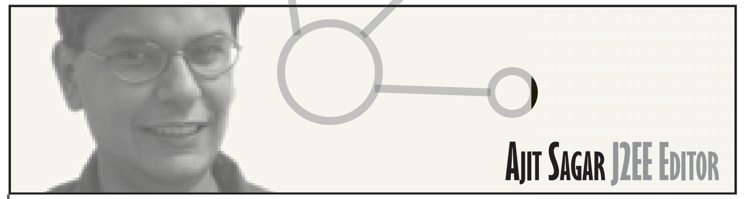
<alan>: Tell us a little about Apache. Is it an official company/body? Do they have offices? Is there a telephone somewhere on the planet with someone answering "Good Morning, Apache, how can I help you?"
<jason>: Apache is a nonprofit corporation, with a board, chairman, president, and assets (servers, trademarks, etc.). There are no central offices. As for a number to dial, we prefer e-mail to phones! For more information, here's what we put at the end of press releases:

> The Apache Software Foundation provides organizational, legal, and financial support for the Apache open-source software projects. Formerly known as the Apache Group, the Foundation incorporated as a membership-based, not-for-profit corporation to ensure that the Apache projects continue to exist beyond the participation of individual volunteers, to enable contributions of intellectual property and financial support, and to provide a vehicle for limiting legal exposure while participating in open-source projects. For more information on the Apache Software Foundation, please see www.apache.org/.

<alan>: Finally, now that there's been time for the news to sink in, what has been your feedback from the community at large?
<jason>: I find people are still asking questions. The issues are very complicated, and the deeper you dig, the more details you find. Eventually you wish you hadn't ventured so deep! Sometimes I wish I'd taken the blue pill. Happily, things appear to be improving, and I hope Q&A pieces like this help people understand what's changing and what it means.

Metrowerks Corp.

www.wireless-studio.com



Ajit Sagar J2EE Editor

Integrating Development

Some years ago I did all my coding in vi, then later in Emacs. I still believe these are great editors; I just don't use them anymore for Java development, especially J2EE application development. I'm much more productive if I use an IDE whose sole purpose in life is to facilitate product development. I can probably still write code faster if I use vi. However, I doubt I could meet my deadlines if all I had was a tool that was primarily meant to be an editor. Emacs is a great environment for setting up and using a development environment. However, it is for all purposes a code editor, not an IDE.

Developing distributed applications is a complex process, and developing the components that make up the application requires coordination and integration of different design and development environments. The software development cycle for the release of a product warrants the use of data and component modeling tools; sophisticated code editors; a fully integrated compilation and build environment; deployment tools; fully integrated debugger, unit, and integration testing environments; and source code version control. Nowadays, you have two options when developing J2EE software. One, use your skills to create a development environment that can use tools from different vendors cohesively and effectively. Two, buy something that provides this environment out of the box.

Which option would you prefer? If you're the only one developing the modules, you may have the luxury to go with the first option. However, if you're involved in the development of J2EE applications, you're much better off with a true integrated development environment that allows you to develop components autonomously, unit test them, test them in your application framework via integration testing, and deploy them into your runtime environment. That way you can easily share your development efforts with your team.

In addition, several IDEs offer refactoring capabilities that will allow you to modify component design after the code has been developed. Several IDEs in the market offer a large part of this functionality out of the box. I've worked with Borland's JBuilder and IntelliJ's IDEA. Both are excellent IDEs that support the development features mentioned earlier, including deployment to the main J2EE application servers, unit testing using JUnit, support for refactoring, and integration with source control systems. There are several other J2EE IDEs to choose from (e.g., WebGain's VisualCafé). Usually application server vendors' tools don't have the level of sophistication offered by J2EE IDEs.

Of course, cost becomes a factor. Some of these IDEs are fairly expensive and buying enterprise licenses can be a costly proposition. However, for large enterprise projects, the benefits accrued by saving precious development hours will be well worth the cost.

What role will the popular editors like Emacs and vi have in J2EE development? Well, some of the IDE vendors are incorporating these into their tools. For example, Java Development Environment for Emacs (JDEE) is a software package that interfaces Emacs to command-line Java development tools (e.g., JavaSoft's JDK). Similarly, TogetherSoft is incorporating vi and Emacs into their tools offerings.

Speaking of development tools, one of the requirements for any successful J2EE project is the right references – online and printed material that gets you the background information you need to get up to speed and hit the ground running. Recently, I came across a new style of books that has proved very useful in developing J2EE software. They're printed as a suite of "CodeNotes" from Infusion Development Corporation – guidelines from developers who have obviously spent some time in the trenches. I've looked at the J2EE and XML CodeNotes, and these are definitely handy references to keep at your desktop.

ajit@sys-con.com

AUTHOR BIO

Ajit Sagar is the J2EE editor of JDJ and the founding editor of XML-Journal. A lead architect with Metavonni, LC, based in Dallas, he's well versed in Java, Web, and XML technologies.

J2EE INDEX

Sun Microsystems

www.sun.com/forte

Infragistics, Inc.

www.infragistics.com

Infragistics, Inc.

www.infragistics.com

Combining RMI with SOAP

Experimenting with SOAP

WRITTEN BY
SAMUDRA GUPTA

At the end of last year, I was given a rather unpleasant assignment. This company had several Java Remote Method Invocations (RMI) services that were interacting with the legacies of the organization and I needed to open up an XML interface for them.

My first idea was to use the Simple Object Access Protocol (SOAP). But the current W3C specification and implementation of the SOAP API from Apache presented only the HTTP binding for SOAP. Because of this, the SOAP services had to be deployed in a Web server container. The possibility of opening up a SOAP-based interface to the existing RMI services was limited by the following factors:

- If we convert the existing RMI services as SOAP-based and deploy them in a Web server container, the existing clients to those services have to be rewritten or reconfigured in order to be able to talk to the SOAP server in a SOAP-recommended way. In the worst-case scenario, we can maintain two interfaces (RMI and SOAP) for the same set of services, which is not desirable.
- The services were interacting with many legacies and in turn with the native programs, which could not be accessed from the Web server system for various security reasons.

We could have implemented a proprietary XML-based solution to overcome the problem. I chose to use SOAP because it's slowly emerging as the new standard for XML-based distributed computing. Moreover, by following a standard, it's easier for other applications to use the services without any particular knowledge of their internals.

In this article, we want to develop a framework that SOAP-enables the existing RMI services by keeping the interfaces intact so they can still be used as normal RMI services. In addition, the framework will demonstrate how to write a SOAP message handler that will receive a SOAP request document, parse it to invoke the requested service, and construct a SOAP response document. We'll also develop a mechanism to specify, deploy, and configure the services with the help of a deployment descriptor file. Then we'll build a client program to access these services.

The framework will be developed by defining a very simple RMI service; we'll be using the Apache SOAP implementation (version 2.2) to build a SOAP client to use the RMI services. *Note:* This framework is just one approach that can be used to talk SOAP over RMI. (The source code can be downloaded from the ***JDJ*** Web site, www.sys-con.com/java/sourcec.cfm.)

Anatomy of SOAP

SOAP provides a simple and lightweight mechanism to exchange structured information in a decentralized and distributed environment. In essence, it's a model for encoding data in a standardized XML format for use in a variety of situations, such as messaging and remote procedure calls (RPC). SOAP consists of three essential parts:

1. ***Envelope:*** This is the top-level XML element or the root element in an XML-encoded SOAP message. The Envelope may contain any or all of the following information: the recipient of the message, the content of the message, and the processing instructions for the message.
2. ***Encoding rules:*** These specify the way the application-defined data-type instances will be exchanged.
3. ***RPC:*** These define a convention for representing the remote procedure calls and the responses to them.

To understand the three parts of the SOAP message let's look at Listing 1.This listing shows a typical SOAP message in an HTTP request binding. The Envelope element is namespace qualified (SOAP-ENV) in order to separate it from any other application-specific identifier. It also contains information about namespace encoding. The immediate child element of the envelope in this example is the Body element. This element *must* be present in a SOAP message; it holds the information about the name of the remote procedure (GetLastTraderPrice) to be invoked and also encodes the parameters required by the remote procedure. In this particular message the remote procedure requires the name of the company (DIS) from which to retrieve the last trade price. Optionally, the Body element contains information about any error that occurred during the RPC and is encoded in a Fault element.

RemoteApps

http://portal.remoteapps.com

The Fault element contains the error/status information with a SOAP message and, if present, can appear only once as a child element of the Body element.

In complex cases, the Envelope element may contain another child element named Header. If this element is present in a SOAP Envelope, it must be the immediate child of the Envelope element. The Header element can typically hold information regarding authentication, transaction management, and more. In our particular example, we would not be using this header information.

A SOAP response document contains a similar structure (see Listing 2). The Envelope element contains the Body element, which in turn contains the requested information, or a fault string, should there be any fault generated during the service call.

For the purpose of this article we'll develop a remote procedure call framework to invoke RMI services, and we'll be using and dissecting the three parts of the SOAP specification.

Conceptual Framework for Sending SOAP Messages over RMI

The basic idea of how to achieve SOAP over RMI relies on the fact that if we can send and receive SOAP request and SOAP response documents over the wire using RMI protocol, we'll be able to achieve our goal. In the process, we'll create a SOAP Envelope document, serialize it, and send it to an RMI service that will decode the SOAP Envelope message, invoke the specified method on the specified Remote object, construct a Response object containing the return value or a fault string in case there's a problem, and again serialize it back to the caller (see Figure 1). The caller then parses the received SOAP response document and processes the result or the fault string accordingly. As the Apache SOAP implementation defines and implements the SOAP-HTTP binding, the objects provided in the API are not serializable in the context of Java RMI, and there lies the challenge – to overcome the present limitation until a more standard implementation of SOAP-RMI comes up.

The Components

Since the requirement is to continue using RMI services and to add a SOAP interface around them, we need to build a SOAP wrapper around the existing RMI services. Also, we need to build a SOAP client following the Apache SOAP implementation. But at the moment the Apache SOAP implementation handles HTTP binding only, so we need to extend it so it can call RMI services. Also, on the RMI server side, we need some mechanism to configure, deploy, and load these SOAP-aware services. Remember, not all the RMI services may need a SOAP interface, so you should be able to specify and configure the services that are required to be SOAP-enabled. Thus, the following components need to be developed:

- SOAP wrapper
- SOAP service manager
- Configuration file – the deployment descriptor
- Apache SOAP extension (this will be a custom Call object, as we will see)
- SOAP client

The SOAP Client

It's unusual to start with the last component specified in the list, but it helps me explain the flow better. As mentioned earlier, we'll be using the Apache SOAP API version 2.2 to write a SOAP client. It's beyond the scope of this article to provide a complete introduction and tutorial of the API, but I'll scheme through the necessary sections as and when required.

First, it's important to understand that to invoke a remote service deployed as a Web service over HTTP or to invoke a remote service over RMI protocol, as we'll be doing, we're essentially making a remote procedure call (RPC). In an RPC, we make a call to the remote procedure and receive a response back. The API provides us with a call and a response object. Also, we often need to pass a few parameters to invoke the remote procedure, and these parameters can be encapsulated in the Parameter object.

The main task is to provide the Call object with the Universal Resource Locator or the URI of the remote service, the name of the method to be invoked as a part of the remote call, and the parameters required by the remote method. The parameters may frequently be complex user-defined data types and objects such as a Person or an Address object. It's also required to define a serialization and deserialization mechanism for these complex objects. By default, the API provides default serializers and deserializers for data types such as Boolean, Double, Float, and Vector. If the data type is complex and user-defined, as an implementer of the SOAP client we have to define our own serializer and deserializer and register it with SOAP by setting a SOAPMappingRegistry for the particular data type.

The term *serialization/deserialization* as used in the SOAP context is different from the serialization notion of Java RMI. In the SOAP context, serialization in a broader term means the XML encoding and decoding of the data types. Again, our example focuses on how to implement these remote procedure calls over RMI and use simple data types to avoid extra work in defining serializers and deserializers for complex data types.

The Call object implemented in the API is tailored to the HTTP binding of SOAP. However, as explained earlier, we want to perform the RPC over RMI. Thus it's required to customize the Call object and extend it as RMICallObject and build the mechanism to achieve the RPC over RMI.

The Envelope object in the API is not serializable to perform an RMI operation. Hence the first step in our framework will be to reproduce a SOAP Envelope element in the form of an XML Document object, which is serializable in the RMI context.

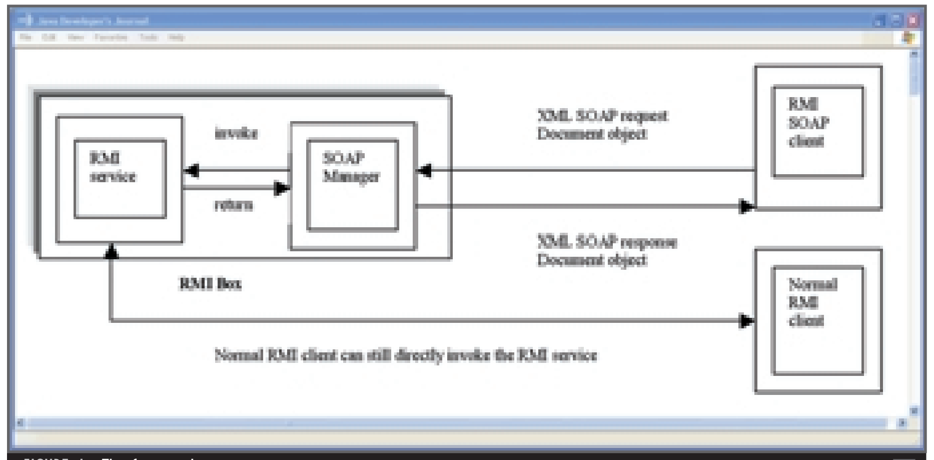

RMI service
invoke
return
SOAP Manager
RMI Box
XML SOAP request Document object
XML SOAP response Document object
RMI SOAP client
Normal RMI client
Normal RMI client can still directly invoke the RMI service

FIGURE 1 The framework

Sitraka

www.sitraka.com/jclass/jdj

```
public class RMICallObject extends
Call
{
public Response invoke(URL url,
 String s) throws SOAPException
private Document buildDocument()
 throws IllegalAddException
private Response buildResponse
      (Document doc) throws
  Exception
}
```

This class structure represents the RMICallObject class that's designed to handle the RMI call. The class overrides the invoke() method from the Call class and defines two private helper methods to construct the Document object, which will be serialized over the RMI call, and another method to construct a Response object out of the returned Document object from the RMI call. For ease of use, I've used the JDOM API from Apache for XML data handling.

The invoke() method takes the URL of the configuration file where the location and name of the RMI server is specified and looks up and obtains a remote reference of the same. As we'll see later, this RMI service will act as an entry point to all other RMI services we want to access (see Listing 3).

SOAPRMIInterface is the remote interface that the RMI service implements in order to parse the Document object that's passed as a result of the SOAP call. The implementation of the SOAPRMIInterface is given a binding in the RMI registry under the name assigned through the variable serverName.

A Document object is constructed out of the generated Call object, which represents the SOAP Envelope.

```
    //construct a Document object
         from the Call object entries
    doc = buildDocument();
```

After obtaining a remote reference of the RMI service, execute the remote method named "invokeMethod()" on this reference by passing the Document object to it. This method is responsible for parsing the Document object and acts as the gateway to all the SOAP RMI services.

```
       //calling the invoke
         method
       responseDoc =
         obj.invokeMethod(doc);
```

Once the Document object is handed over to the remote service, the client-side process waits for a return Document object from the RMI server. The remote server object processes the passed Document object.

> I chose to use SOAP because it's slowly emerging as the new standard for XML-based distributed computing

Once we've extended the Call object to the RMICallObject and incorporated the functionality to perform an RMI call, the rest of the client programming follows what we do for a normal SOAP client (RMIClient.java) (see Listing 4).

This listing is an example of the SOAP client. First, we obtain the reference to an RMICallObject, then pass the target remote service name as it's bound to the RMI registry, pass the name of the remote method to be invoked, create the Parameter object to pass the required parameters to the remote method, and, finally, invoke the remote method by calling the invoke() method on the RMICallObject. It's important to note that we pass the URL as the address of the configuration file (conf.txt) that holds the name and location of the SOAP server manager. As shown in Listing 4, the conf.txt file should be placed under the current working directory.

Once this URL is passed to the invoke() method of the RMICallObject class, it constructs a Properties file out of the given configuration file conf.txt and an RMI lookup URL with the server name and the location.

Once the returned XML Document object is obtained from the RPC, we can analyze the entries of the Document to obtain the returned result of the method call or the fault string associated with this method call, and construct the Response object out of that to return it to the Caller RMI client.

SOAP Wrapper

A major part of building this framework is to develop a SOAP wrapper around RMI services, which can parse and handle incoming SOAP requests. So, the basic idea of this framework is to serialize the SOAP request over RMI in the form of an XML Document object, and the SOAP wrapper is required to be an RMI service that can accept the serialized Document object. As a matter of fact, this is an RMI service acting as an entry point to all other RMI services deployed as SOAP-enabled, so it's vital to ensure that this service is up and running so other services can be accessed and used.

First, declare a Remote interface for this RMI service.

```
public interface SoapRMIInterface
  extends Remote
{
 public Document
   invokeMethod(Document doc) throws
   RemoteException, RMISoapException;

}
```

Our SOAP wrapper RMI service will implement this interface, and, as a result, define the invokeMethod() method. As we can see, apart from throwing the usual RemoteException, this service also throws a customized RMISoapException to differentiate between normal RMI failures and any SOAP-related ones. This design is guided by the fact that to build a Response object, we need to pass the proper fault string information back to the caller object.

The implementation of the invokeMethod() method must perform the following tasks:

- Parse the received document and retrieve the Body element of the SOAP request.
- Identify the name of the remote service, the name of the remote method, and the parameters passed to it.
- Locate the remote service from the registry by analyzing the deployment descriptor (to be discussed in the following section).
- Check whether the method specified is exposed as a part of the given service and if the service is defined in the deployment descriptor. If it can't locate the specified service, it throws an exception and eventually propagates it as a SOAP fault string back to the caller.

AccelTree

www.acceltree.com

- Create a new instance of the specified service implementation class and invoke the specified method by passing the parameters obtained as a part of the SOAP request, if the method specified is a part of the deployed service. If the method name could not be located, it throws an exception and propagates it back to the caller as a SOAP fault string.
- Create and return a response XML document to the caller as an RPC response once the service is successfully invoked. After receiving this XML document, the RMICallObject then constructs the Response object out of it and passes it to the SOAP client object.

This process is large in scope; the source code for SoapRMIServer is available on www.sys-con.com/java/sourcec.cfm. To check if the method exists for a given service, the service uses the Java reflection mechanism. It also uses this mechanism when it constructs a Response XML Document object out of the returned value of the service, in case the returned value is an object. While creating a response XML document to maintain the SOAP response specification, it uses the received SOAP request document to maintain the integrity of the namespace and encoding style used.

“SOAP provides a simple and lightweight mechanism to exchange structured information in a decentralized and distributed environment”

The Deployment Descriptor

SOAP utilizes an XML file called *deployment descriptor* to supply information to the SOAP runtime environment. The deployment descriptor contains the Universal Resource Name (URN) for the service, the name of the method, the Java implementation class if the SOAP service is deployed as a Java class, and more regarding serialization and deserialization (in a SOAP context and not in an RMI context). The amount of information that can and should be supplied depends on the SOAP runtime requirements. In the example we're following, which is a simplistic model, we need information regarding the RMI services that are deployed as SOAP-enabled services.

Typically, our deployment descriptor(s) will contain the following information:

- The URN of the service as the name that binds it to the RMI registry
- The name of the Java class deployed as the specified service
- The method that's exposed as the interface to the service
- Whether or not the method is implemented as static

The other bits of information that normally become a part of the SOAP service, such as the scope of the service (application/session, etc.), are not relevant to the framework we're developing, because in this particular framework I've tried to avoid the complications of session management.

The deployment descriptor is typically encoded within a top-level element <isd:service>, which is namespace-qualified to avoid conflict with any other user-defined entity. An example of a deployment descriptor is as follows:

```
<isd:service xmlns:isd=
   "http://xml.apache.org/xml-
     soap/deployment" id="urn:greet
     ingserver">
  <isd:provider type="java"
     scope="Application"
     methods="sayHello">
     <isd:java class="Greeting" stat
        ic="false"/>
  </isd:provider>
</isd:service>
```

This descriptor particularly defines a service named “greetingserver” with an exposed method “sayHello”, and the implementation Java class of the service is “Greeting” and the method is nonstatic.

The basic idea is to provide an XML file in which we can describe the deployed services. To deploy all the services in a single file, I've put the individual descriptors under a root element called “<soapservices>” (soap.xml).

The Apache SOAP implementation provides a utility class called DeploymentDescriptor to load and parse the deployment descriptor files. I've used the same to load and access my consolidated deployment descriptor file, along with a ConfigDescriptor class designed to parse the file and retrieve everything or a descriptor for a particular service against its service ID.

The Service Manager

So far on the server side we've created the SOAP wrapper RMI service, which is the gateway to all other SOAP-enabled RMI services, and have also devised a deployment descriptor to specify and describe all the services that are to be deployed and loaded as SOAP-enabled. As a precondition to these specific services working, we need to load all the services specified in the deployment descriptor file along with the wrapper RMI service and bind them to the RMI registry. Binding is straightforward. The only difficulty I faced was in using the DeploymentDescriptor class of the Apache SOAP API, as it doesn't accept the fact that more than one service can be specified within the same descriptor file.

I designed a ConfigDescriptor class (ConfigDescriptor.java) that parses the deployment descriptor XML file and creates an individual DeploymentDescriptor object by taking each child node of the <soapservices> root element. I added a few convenient methods to retrieve all or one particular DeploymentDescriptor object given the service ID (the URN) of the service. The SoapServerManager class uses the ConfigDescriptor object and retrieves all the DeploymentDescriptor objects specified in the descriptor file. It then finds all the Java classes specified as the service implementation and binds them to the RMI registry with the names specified in the ID attribute of the <isd:service> element.

This brings all the specified RMI services up and running, with all the methods specified in the deployment descriptor file exposed as SOAP-accessible remote service methods. Once this is done, the last job of the SoapServerManager class is to load the SOAP wrapper RMI service. The additional functionality of starting up an individual service or taking down one or all of the services can be provided through the SoapServerManager class.

Macromedia

www.macromedia.com/go/jrun4jdj

A Simple RMI Service

Consider a simple RMI service. This service is named "addressserver" and the remote service method is "getAddress", which takes two parameters – the name and surname of a person. Given the name and surname, the service returns the person's address. To make it a very small example service, I've hard-coded the address inside the code, although in real life this service might interact with a database and fetch the address. I've created an Address object that holds the name, city, and zip code. The service returns the Address object in response to a SOAP call (AddressService.java) (see Listing 5).

This listing explains the structure of the addressserver service, which is also designed as an RMI service.

Running the Server

To run the server, we need to ensure the following:

- Place all the .jar files (mail.jar, xerces.jar, activation.jar, soap.jar, and jdom.jar) under the lib directory of your install directory.
- Set the CLASSPATH to include the current working directory and the lib directory under it.
- Generate the stubs and skeletons of all the RMI services to be deployed (included in the source code). For the SoapRMIServer, it's important to provide all the .jar files in the rmic –classpath option.

```
rmic -classpath .;c:\your-
  directory\lib\jdom.jar; c:\your-
  directory\lib\soap.jar; c:\your-
  directory\lib\xerces.jar; c:\your-
  directory\lib\mail.jar; c:\your-
  directory\lib\activation.jar
  rmi.server.SoapRMIServer
```

- Run the Service Manager (SoapServerManager). This will read the deployment descriptor file, load all the specified RMI services there, and bind them to the RMI registry under the name specified as the service ID.
- Provide the deployment descriptor in the CLASSPATH.
- Check your security policy for problems starting up or accessing the RMI services. For this example, I created and supplied a policy file (myPolicy.txt) that grants all permissions. If you create/use this policy file, pass it to the runtime with the following command:

```
java - Djava.security
   .policy=myPolicy.txt
   rmi.server.SoapServerManager
```

Running the Client

The client program, as explained, is ready to call the deployed addressservice. We can run the client program using the following command from a DOS prompt – java rmi.client.RMIClient.

```
The address of the rmi server:
rmi://localhost:1099/soapserver
Target obj: urn:addressserver
The return value:
person...Samudra Gupta
city...London
post...SS1 2RQ
```

This shows the result of the client call and we receive the address of the specified person.

To find out how the framework behaves in case of erroneous inputs, I changed the method name to "getLocation" instead of the correct exposed method "getAddress, and received the following error message:

```
The address of the rmi server:
rmi://localhost:1099/soapserver
Target obj: urn:addressserver
The fault code: SOAP-
ENV:Server.BadTargetObjectURI
The return value:
faultcode...SOAP-
ENV:Server.BadTargetObjectURI
faultstring...Could not invoke the
specified method
```

It's important to note that the client will also need all the .jar files to be set in the CLASSPATH as they're used by the Apache SOAP implementation as well as our own extended RMICallObject class.

Conclusion

This framework served my purpose very well, and I could use the existing RMI services both as normal RMI as well as SOAP deployed over RMI. This framework is tailored to meet my requirements, so it may not meet yours. The ability to handle a user-defined data type as a parameter to the remote services is surely one of the main areas to be incorporated; although it's used in the real project, I didn't explain it here due to space constraints. I strongly believe that more experimentation with SOAP will eventually help it become a mature and robust technology.

Resources

- *Apache SOAP API for an API download, mailing list, and FAQ:* http://xml.apache.org/soap/index.html
- *An introduction to RMI:* http://java.sun.com/docs/books/tutorial/rmi/index.html

samudrag@hotmail.com

Author Bio
Samudra Gupta is an independent Java consultant in the UK, providing solutions for e-commerce projects. He has five years of experience in Java and Web technology.

Listing 1

```
POST /StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
SOAPAction: "Some-URI"

<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
   <SOAP-ENV:Body>
       <m:GetLastTradePrice xmlns:m="Some-URI">
           <symbol>DIS</symbol>
       </m:GetLastTradePrice>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Listing 2

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
   <SOAP-ENV:Body>
       <m:GetLastTradePriceResponse xmlns:m="Some-URI">
           <Price>34.5</Price>
```

Compuware Corp.

www.compuware.com/products/optimalj

```
      </m:GetLastTradePriceResponse>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Listing 3

```
//construct a Properties object and load with it the properties
file specified by the URL
         Properties prop = new Properties();
         prop.load(url.openConnection().getInputStream());
         //construct a Document objet from the Call object
             entries
         doc = buildDocument();
         //obtain the name and address of the rmi server frm
             the properties file
         serverURI = prop.getProperty("serverLocation");
         serverName = prop.getProperty("serverName");
         System.out.println("The address of the rmi server:
             "+"rmi://"+serverURI+"/"+serverName);
         //get the rmi server URI
         System.out.println("Target obj:
             "+this.getTargetObjectURI());
         //obtain the reference to the rmi server object by
             the target object URI
         SoapRMIInterface obj =
         (SoapRMIInterface)java.rmi.Naming.lookup("rmi://
             "+serverURI+"/urn:+serverName);
```

Listing 4

```
//create a RMICallObject
         RMICallObject call = new RMICallObject();
         //set the target remote service name
         call.setTargetObjectURI("urn:addressserver");
         //set the name of the remote method to be invoked
         call.setMethodName("getAddress");
         //set the encoding style to be used
         call.setEncodingStyleURI(Constants.NS_URI_SOAP_ENC);
         //creating the parameters to be passed to the remote
             method
         Vector params = new Vector();
         params.addElement(new Parameter("name1", String.class,
             "Samudra", null));
         params.addElement(new Parameter("name2", String.class,
             "Gupta", null));
         call.setParams(params);
         //obtaining the current working directory
         String workingDir = System.getProperty("user.dir");
         String url =
             "file://localhost/"+workingDir+"/conf.txt";
         System.out.println("Invoking..."+url);

         //invoking the remote service
         Response res = call.invoke(new URL(url), "");
```

Listing 5

```
public Address getAddress(String personName, String surName)
     {
         Address add = new Address();
         StringBuffer buffer = new StringBuffer(personName);
         buffer.append(" ");
         buffer.append(surName);
         add.setPerson(buffer.toString());
         add.setCity("London");
         add.setPost("SS1 2RQ");
         System.out.println("Returning address for:
             "+add.getPerson()+" "+surName);
         return add;
     }
```

Download the Code!
www.JavaDevelopersJournal.com

INT, Inc

www.int.com

SilverStream Software

www.silverstream.com/scale

Spread Your Wings with Cocoon

A powerful tool for site designers

By David Rosenstrauch

Apache Cocoon is one of the most interesting, innovative, and powerful platforms for dynamic content generation, though not as well known as the others. A subproject of the Apache XML project, Cocoon is one of the lesser-known offerings from the folks at the all-open-source Apache Software Foundation, having garnered less attention than some of its more popular cousins like Struts. But Cocoon is worth a look.

It's not just Cocoon's use of XML in content generation that makes it so interesting; it's *how* it uses XML. Cocoon's authors clearly have a deep experience with and an understanding of XML – what it is and isn't good for – and Cocoon's simple but powerful architecture reflects that experience. XML isn't used here just "because everyone's using it." Rather, Cocoon exploits XML's strength for separating content from presentation. (As we know now, the lack of that separation made it increasingly difficult to do Web page development in straight HTML.) The result is an innovative and powerful tool for content site developers.

This article will familiarize you with Cocoon and some of its related technologies: what it is, what it does, and how to start using it in your own development projects.

A basic understanding of the core concepts behind XML, SAX, and XSL (and, of course, HTML) is helpful when reading this article. Don't worry too much if you haven't worked with these technologies, though. I won't be delving too deeply into them and will try to make any examples easy to understand.

What Is Cocoon?

Although at heart Cocoon is "yet another dynamic content-generation platform" (technically putting it in competition with the many other content-generation technologies out there such ASP, JSP, PHP, and Struts), Cocoon adds some new twists to this category that make it stand out.

Cocoon's core difference is its use of XML throughout the content-generation process. Each request sent to the Cocoon framework is processed using the same three steps:

1. Generate XML content (either statically or dynamically)
2. Optionally transform it
3. Format it for output

/n software inc.

www.nsoftware.com

Along with this simple, easy-to-understand architecture comes great power and flexibility for developers. Although Cocoon is most often used for generating Web pages, it's by no means limited to that. It can generate any type of output you desire for any type of client device you like: HTML, XML, text, WML for WAP-enabled devices such as mobile phones, SVG images, Postscript, Adobe PDF, etc. And, of course, you can "roll your own" add-ons for Cocoon to generate any type of custom output format you need.

Another major plus of Cocoon's XML-orientation: it provides for excellent separation of content and presentation – that holy grail of software applications. Content is kept as presentation-free XML data for as long as possible during processing, and then formatted into the appropriate output format just before being returned to the user.

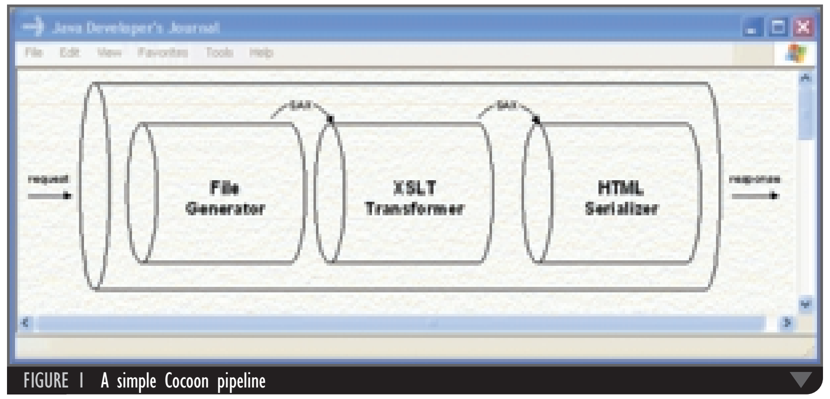


FIGURE 1 A simple Cocoon pipeline

In fact, Cocoon strives for an even greater separation of concerns. Its philosophy is to look upon the process of content generation as three separate realms: content, logic, and style. This type of division makes a great deal of sense, especially when you consider that completely different teams of people are frequently assigned to each of these functions: logic to software developers, content to users and data entry staff, and style to graphic designers.

How Cocoon Was Hatched

Inception

Cocoon began its life in 1999 as a far less ambitious endeavor than it is now. Pioneered by Apache developer Stefano Mazzocchi, Cocoon was initially just a "proof of concept" – a servlet that used XML and XSLT transformations to generate its output.

Cocoon 1.0

By the time Cocoon made it to version 1.0, it had progressed into a full-fledged framework for XML content generation and was starting to receive a good deal of recognition and use by site developers.

As with all early-version software though, it's often difficult to foresee the potential usability problems that will crop up in practice while the software is still being developed. It's also difficult to envision how popular your application will be at such an early stage. Cocoon was no different. Version 1.0, although functional, had its usability hampered by design decisions made early on, most notably its reliance on the memory-intensive XML DOM architecture. (The SAX model, and the APIs and tools needed to use it, were still in their infancy at that point.)

Since Cocoon was proving to be quite popular, demands for new features and improved performance kept coming in. It soon became clear that the initial architecture was not adequate to address these issues.

Cocoon 2.0

Enter Cocoon v2.0. This version (released as alpha in March 2001, with the first production release completed in November) seems to be almost a complete rewrite of the application. It addresses the performance issues from version 1 as well as being a much cleaner architecture conceptually.

The first notable improvement is the substitution of the event-driven SAX XML standard for the memory-intensive DOM API. In addition to the improvements in memory efficiency and scalability, the SAX model also allows output to be generated incrementally. This provides a faster response time since a response page is returned little by little, rather than waiting until all processing is complete to return a page (as the DOM model required).

The second major improvement concerns the internal architecture of the Cocoon application. Originally structured using a Reactor design pattern, this structure apparently caused conceptual as well as implementation difficulties. Instead, version 2.0 substitutes a pipeline architecture (described later) that proved far more flexible to code as well as much clearer conceptually.

The result is a solid, well-tested, powerful, and more efficient framework for just about any type of content generation under the sun. At the same time it manages to elegantly achieve true separation of presentation and content.

In short, Cocoon really rocks!

How Does Cocoon Work?

Let's take a closer look at Cocoon and how it works, and see how you can put it to use in your own development.

A Servlet at Heart

Although Cocoon is a powerful framework for XML processing, it's just a servlet at heart. Its job is just like any other servlet's: to receive requests, process them, and then generate a response. Cocoon accomplishes this by taking each request, finding an appropriate "pipeline" to handle it, executing the pipeline, and returning in its response any output that the pipeline generated. The pipeline's function is to generate the response output for a particular request, using XML processing internally to accomplish this task.

The Pipeline Architecture

The pipeline, a simple and elegant paradigm, fits in extremely well with the XML SAX processing model.

A pipeline at its simplest consists of a sequence of the three core Cocoon components – generators, transformers, and serializers – arranged in a chain (see Figure 1). XML data (SAX events) is passed down the chain, with each component performing its own processing on the data as needed. At the end of the chain the events are serialized out to the response's OutputStream and returned to the client making the request.

Generators, Transformers, and Serializers

The first component in the chain is always the generator. The generator's job is to create the stream of XML events that will be fed through the rest of the pipeline. There are prebuilt generators available to create the XML events from a number of possible sources: an XML file on disk, an HTML file (the HTML is tidied up and turned into XHTML in order to be XML-compatible), a JSP page, an XSP page (more about XSP later), etc. In fact, there are over a dozen varieties of generators included with the Cocoon distribution. And you can easily create new ones if you need to generate events from a nonstandard source.

Canoo Engineering AG

www.canoo.com/ulc/

"The result is a solid, well-tested, powerful, and more efficient framework for just about any type of content generation under the sun"

The last component in the chain is always the serializer. The serializer's job is to turn the stream of XML events into some form of output that will be returned in the response. Prebuilt serializers are available to create output in the most popular formats: XML, HTML, text, WML, an SVG image, and more. Again, over a dozen varieties of serializers are included with the Cocoon distribution and again you can easily roll your own to support just about any output format you like.

As an option, a sequence of one or more transformers can lie in between the generator and the serializer. Transformers allow the developer to manipulate the XML events coming down the pipeline – adding, removing, or modifying events as needed – before the serializer finally sends them back in the response.

The XSLT transformer is the most common – and most powerful – transformer. It runs an XSL stylesheet against the stream of XML events coming down the pipeline, allowing the developer to use the powerful XSLT language to transform the XML from pure data into styled output.

You can place multiple transformers in a row in the pipeline, each of which will operate on the XML events one at a time. This allows you to style the data incrementally, and can help keep your stylesheets smaller and simpler.

Although Cocoon uses several other types of components as well (which are beyond the scope of this article), these three components are the core of its architecture. Pretty simple, huh? But it sure is powerful! By assembling combinations of these core components – along with your own custom-built server pages and stylesheets – you can build pipelines to generate content from any data source you like, styled however you like, and rendered in whatever output format you like.

Putting It All Together

Let's look at a sample "Hello World" pipeline and see how this all ties together in practice.

Our "Hello World" pipeline will work as follows:

- Use the file generator to read XML from a file HelloWorld.xml
- Use the XSLT transformer to run a stylesheet Style.xsl against the XML data and translate it into formatted HTML
- Use the HTML serializer to return the resulting HTML page to the user

Let's start by looking at the files mentioned earlier:

HelloWorld.xml

```
<?xml version="1.0"?>
<message text="Hello World"/>
```

The HelloWorld.xml file is very simple, consisting of a single node (<message>) that contains a single attribute (text).

The Style.xsl stylesheet (see Listing 1) is also very simple, consisting of only two formatting transformations. The first one (xsl:template match="/") is called when the XSLT processor begins processing the document. It generates the skeleton of an HTML page. The body of the page is left empty, however, except for the XSL instruction xsl:apply-templates. This instruction simply commands the XSL processor to begin processing any child nodes here, applying other templates as needed. The net effect of the instruction then is "transform any child nodes here."

In this case there's only one node in the XML file, a <message> node, and only one remaining template in the stylesheet (xsl:template match="message"), which is looking to match <message> nodes. Since the stylesheet's template matches the XML file's node, we'll perform the second transformation:

- Write a <h1> opening tag
- Write the value of the text attribute in the <message> node (in this case "Hello World")
- Write a </h1> closing tag

Note the conceptual separation of concerns between the XML file and the XSL stylesheet. The XML file contains data only – the text of a message – with no indication as to how it should be formatted for output. The stylesheet, on the other hand, contains formatting only – instructions on how a text message should be displayed – that can be applied to any XML data containing text messages. As a Cocoon developer, you'll get the most out of the framework if you structure your sites in this fashion.

Once Cocoon executes this pipeline and performs this transformation, the result is the following HTML:

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
<title>A Message From Cocoon</title>
</head>
<body>
<h1>Hello World</h1>
</body>
</html>
```

This HTML is then serialized using the HTML serializer (which takes care of the few incompatibilities between strongly formatted XML and loosely formatted HTML) and the whole thing is sent back in the response to end the request.

The Sitemap

Great! So how do we write the code for this pipeline?

Cocoon uses a file called *sitemap* to define all the pipelines in your application. The sitemap is just that, a map of your Cocoon Web site. It defines which pipeline will be run in response to each site request, and how exactly each pipeline will generate its response page.

The sitemap is written in, guess what? XML, just like everything else in Cocoon. Let's look at it piece by piece.

First, all sitemaps must contain the <map:sitemap> root element:

```
<?xml version="1.0"?>
<map:sitemap
 xmlns:map="http://apache.org/cocoon/sitemap/1.0"
>
```

SpiritSoft

www.spiritsoft.com/climber

Then the sitemap lists which Cocoon components your site will use (see Listing 2). In this case we'll be using only three components:
1. The file generator
2. The XSL transformer
3. The HTML serializer

Each of these components is labeled as being the default component of its type (more about this later).

We'll also have to define an additional component, a matcher, to get this sitemap to work. A matcher is used to match the URL that the user enters and route it to the appropriate pipeline. (We won't discuss matchers in this article though.)

Then we define the pipelines used in the site. In this case we have only one, our Hello World pipeline, which we will set up to be executed when a request arrives for page "HelloWorld.html".

The pipeline calls the file generator to read from the HelloWorld.xml file, then calls the XSL transformer to apply the Style.xsl stylesheet, and finally calls the HTML serializer to properly format the XML event stream as HTML.

Since earlier in the sitemap we defined each of these components to be the default of its type (see Listing 2), we can use a shortcut and not explicitly write which component we're using; Cocoon assumes we're using the default. (However, if we were calling a generator other than the file generator, a JSP page, for example, we would need to write something like <map:generate type="jsp" src="HelloWorld.jsp"/>.)

The full pipeline reads like this:

```
<map:pipelines>
 <map:pipeline>
  <map:match pattern="HelloWorld.html">
   <map:generate src="HelloWorld.xml"/>
   <map:transform src="Style.xsl"/>
   <map:serialize/>
  </map:match>
 </map:pipeline>
</map:pipelines>
```

Finally, we write a closing tag for the root element:

```
</map:sitemap>
```

And that's it. Our complete Hello World sitemap reads like Listing 3.

Installing and Running Cocoon

How Do We Run This Site?

As mentioned earlier, Cocoon is just a servlet at heart. It can be easily run on any servlet engine that supports version 2.2 or later of the Servlet API. I've used it with Apache Tomcat, but it can also be run on WebLogic, Resin, and many others, even on Microsoft IIS (using ServletExec).

Installing Cocoon onto the server is pretty easy for most servlet engines and usually consists of the following:
- Download one of the Cocoon binaries (e.g., cocoon-2.0.1-bin.zip) from http://xml.apache.org/cocoon/dist/.
- Unzip the archive.
- Grab the cocoon.war file that was extracted from the archive and copy it to the appropriate directory on your servlet engine (e.g., for Tomcat, the "web apps" directory).
- Restart the servlet engine; it will automatically install cocoon from the .war file when it's restarted.

More details and instructions for specific servlet engines can be found on the installation page at the Cocoon Web site: http://xml.apache.org/cocoon/installing/.

Once Cocoon has been installed, running it is just a matter of accessing a URL that's handled by the Cocoon servlet. When a request to such a URL is made, it is routed to the Cocoon servlet. Cocoon matches the URL against its sitemap and then executes the appropriate pipeline.

To run our Hello World site, we first need to take the sitemap we just wrote and overwrite the sample sitemap.xmap file that Cocoon provides us by default. Then we just point our browser to http://localhost:8080/cocoon/HelloWorld.html and – voilà – Cocoon serves up our dynamically generated "Hello World" page.

The Power of Cocoon

Our "Hello World" pipeline is an extremely simple example. However, it's not hard to see how applying these concepts can enable us to create more complex sites with Cocoon.

Since Cocoon provides complete separation of content from style, you can take the same content and format it in many different ways. There's no need to create new logic or content in order to create different looks for your site. Just create a new stylesheet for each output format, and you can serve up completely different-looking sites from the same content.

How could this be useful in practice? Imagine the following possibilities, all of which can be accomplished with ease using Cocoon. You could create sites that serve out the same content formatted completely differently based on:
- Various devices that clients use to access the site (a PC with a Web browser, a WML-enabled phone, a PDA using the HandWeb browser, etc.)
- Different browsers that clients use to access the site (e.g., display the site differently if the user is using Netscape, IE, or Opera; provide special support for old versions of these browsers; take advantage of the advanced features of newer versions of these browsers, etc.)
- Machine-readability: the site can generate both a user-readable version (in HTML) and a machine-readable version (in XML)
- The user's language (you could internationalize your site using stylesheets)
- Different security levels
- And more

Clearly, Cocoon's ability to do dynamically styled page generation is a powerful tool for site designers!

XSP

Another key innovation to come out of the Cocoon project, which I mentioned briefly above, is Extensible Server Pages (XSP). Inspired by JSP, XSP provides all the power of JSP while removing one of that technology's major drawbacks: the intermingling of content and style.

As discussed earlier, Cocoon heavily stresses the separation of content, logic, and presentation. If there's one place that logic and presentation are often intermingled it's in server pages. By definition, both ASP and JSP freely intermix logic and presentation, i.e., source code and HTML. Although the use of beans and taglibs in JSP can minimize this to some extent, there's still inherently some intermingling of logic and presentation, due to the use of HTML.

Cocoon's solution is, once again, elegant: use XML instead of HTML in your server pages. Unlike HTML, XML is presentation-free; it's just data. So writing a server page using XML makes a lot more sense.

An XSP page therefore consists of XML data tags, along with intermingled logic (Java code). As with JSP, the Java logic

AltoWeb

www.altoweb.com

(through the use of either embedded code or calls to external modules) dynamically creates the page to be output. The difference here is that, once again, presentation-free XML is what the logic will generate, not HTML.

All XSP pages are Cocoon generators – the source of XML events in a pipeline. Once the XSP page has executed and generated the appropriate XML stream, the stream is then typically styled and formatted using a Cocoon transformer (e.g., the XSLT transformer using an XSL stylesheet) into the appropriate output format (such as HTML).

Like JSP, XSP pages are compiled into Java code (and then eventually class files), and like JSP, XSP also provides support for tag libraries (often referred to as "logicsheets" in Cocoon). As JSP developers know, calling reusable tag libraries in your pages helps to keep them from becoming too filled up with Java code. Using tag libraries with XSP provides the same benefits.

XSP is too big a topic to discuss in more detail here. (It could easily fill up an entire article on its own.) This should be a good overview though, and you can refer to the Resources section if you'd like to read more about XSP.

A Cocoon Case Study

I recently used Cocoon to develop a site for a New York City law firm. The project, its design, and some of my reasons for choosing Cocoon are described here to help provide some insight as to when you might want to choose Cocoon as a development platform on your own projects.

"XSP provides all the power of JSP while removing one of that technology's major drawbacks: the intermingling of content and style"

The law firm was looking for a new piece of software to replace the ancient and inflexible software they were currently using and getting increasingly locked into (Microsoft Works…for DOS!). The system functionality was not terribly complex – a basic CRUD system (functionality for create, read, update, and delete) that would provide a user-friendly front end for the legal case files in their database. The entire application would consist of less than a dozen screens.

Although the head attorney was fairly computer-savvy (he had recently mocked up a prototype for the new system in MS Access), he was looking to me for technology recommendations and was happy to defer to my knowledge and experience.

My first recommendation to him was to choose a Web-based system over an application in MS Access. This was an easy decision for me, as there would be several benefits to be gained from a Web-based system, including ease of development, minimal training required for the rest of the staff, and no software installation or upgrade procedures needed.

But which Web development platform to choose was a bigger question. Off the top of my head JSP and Struts were the leading candidates, but I also wanted to consider some newer, more cutting-edge technologies as well. As I had heard of Cocoon before, and had worked heavily with XML on a recent project, I started reading up on Cocoon to see if it would be a good fit.

Cocoon's technology was intriguing, and my experience with XML enabled me to come up with an idea that I realized could save a good bit of development time. Since the screens were fairly simple and similar (just rows of fields from the database) I realized that I could design the GUI extremely quickly and easily by just mocking up the screens in XML (see Listing 4, a mock-up of a Web page in XML that will later be rendered using HTML tables).

Then switching hats and putting myself in "style" mode, I could turn all the screen mock-ups into Web pages just by writing a single stylesheet that would transform the XML into HTML. Each <page> tag could be transformed into a skeleton for an HTML page, each <section> tag could be transformed into an HTML table, and each <row> tag could become a row in the table. The idea appealed to me.

What finally clinched the decision to use Cocoon, however, was an additional requirement, one that I initially wasn't sure how to accomplish. Law firms, as we all know, generate reams of documents, and one of the reasons this firm had stuck with MS Works all these years was its ability to mail merge the information from the database into a document. If they were going to abandon Works, the new system would need to provide mail-merge functionality as well. At first, I had no idea how I would provide a mail merge in a Web-based system. But as I thought it through I began to formulate a plan.

First of all, I realized the best approach would be for the site to serve up the merged documents as a download from a Web page. This would be simple for the users. Most browsers' "open-attachment" functionality is automatically configured to launch the appropriate application when an attachment is opened, so each time the user generated a mail-merge document, the word processor (MS Word) would automatically launch and open to that document. This would work out quite nicely.

But how to do the mail merge itself? Although MS Word has a mail-merge utility, I felt it would be clumsy for the users. It would be much simpler for them if they could just click on a button labeled "create merged document" and have the document arrive with all the merge substitutions already done behind the scenes. What would be required to make this happen?

Having the site retrieve the appropriate data from the database was certainly easy enough. After that, I reasoned, the template document would need to be read in, the merge fields identified, and the actual data substituted in.

Reading an MS Word document was a tall order – I wasn't aware of any Java libraries that could do that. But what about an RTF (Rich Text Format) file? RTF, unlike the Word .doc format, was text-based and would be much easier to read. In fact, I could probably write a parser to do it. I read though the RTF spec and after spending a couple of hours with the JavaCC parser generator, I was able to successfully read RTF documents and find the mail-merge fields in them. I checked with the head attorney to make sure he didn't mind using RTF format instead of Word, and he didn't. As long as they could still use the MS Word application to edit the documents (which they could), he was fine with it.

That left me with the last bit: How to substitute the data retrieved from the database for the merge fields? Boy, I thought, it would be nice if there was some existing code that could already do this so I wouldn't have to write it from scratch. What type of software could I use to scan through a document for a particular piece of content and change the

Rational Software

www.rational.com/offer/javacd2

Author Bio
David Rosenstrauch, a software developer, has been providing development and consulting services to Fortune 500 companies and start-ups for over 13 years.

value of that content before outputting it again? Then it hit me: I could use an XSL stylesheet! XSL was built to easily handle tasks like this.

Suddenly the whole idea began to come together, and my decision to use Cocoon was clinched. I would write a new generator that would parse an RTF file and turn it into a stream of XML events. I would generate a stylesheet in response to each mail-merge request that substituted database values for the merge fields and I'd use the XSL transformer to apply this stylesheet. Then I would use the text serializer to write out the new mail-merged RTF file data, I'd set the appropriate MIME type for RTF documents ("application/rtf"), and the user would get served up a mail-merged RTF document. I put together another proof-of-concept and, sure enough, it worked.

Writing the application using Cocoon worked out well, though it did have its share of challenges. As the application was not particularly complex, the code was not particularly difficult. The biggest challenge, however, was getting up to speed in some of the new technologies I was using, primarily Cocoon and XSL. I wound up learning them incrementally, as needed, when I hit roadblocks in various pieces of the development. ("Hmmm. How do I do *this* in XSL?") The online documentation I found for XSL and Cocoon was helpful, as was Michael Kay's book, *XSLT Programmer's Reference 2nd Edition*. (And, of course, I also posted my share of "Help me!" messages to the Cocoon Users mailing list.)

The system was finally completed and installed in December 2001, and got a big thumbs-up from both the users and the head attorney. It is now used daily by the entire staff.

From my perspective, I give Cocoon a big thumbs-up. I chose it as the core technology for this project, and it accomplished everything I needed. I found that using XML and Cocoon on the project allowed me to deliver it faster, as well as helping out tremendously in the conception and design phase. While designing I was able to focus completely on what type of content I was going to display on each page and how it would be generated, and completely ignore all presentation and style concerns until a later time. I found this separation of concerns during design to be quite a refreshing change!

Cocoon Concerns

Although I found Cocoon to be an excellent technology for site development, it's not without some drawbacks. You should be aware of the following concerns when you're deciding whether to use Cocoon in your development:

- Cocoon is still a fairly new and less established technology. And there's still a risk that it might not catch on and achieve a large user base. This translates into several direct risks for systems built with Cocoon:
  – It's still more difficult to find developers experienced in Cocoon and its related technologies than in more mainstream technologies like JSP or Struts.
  – If it doesn't catch on, it may be difficult to continue enhancing and supporting applications built with it.

- Cocoon, like many new technologies, is still undergoing development. Although most of the major redesign work appears to be behind it, the functionality and APIs may continue to change in the future, forcing developers to update their applications to be compatible with new releases.

- As with many technologies, making life easier for the developer often comes hand-in-hand with a performance penalty. As we know, even our choice of using the Java language comes with a performance hit. But if the hit is minimal enough and the efficiency gains significant enough, the performance issue is outweighed. Cocoon is no different – its power comes at a price. Although many Cocoon components are eventually compiled down to class files (allowing optimizers like JITs and HotSpot to do their magic) all that XML and XSL processing does take a performance toll. (Cocoon does utilize techniques to make this processing as efficient as possible, such as an extensive caching mechanism.) You need to decide for yourself if it's worth it. For a law firm's document processing system I felt that the answer was yes. For building a high-volume site where performance is critical, like Amazon.com or a stock-trading system, your answer might very well be no.

- The sitemap/pipeline examples I gave here were very simple and fit cleanly into Cocoon's generator-to-transformer-to-serializer paradigm. But many sites require more complex functionality than that to respond to a request, such as conditional processing, loops, etc. The sitemap does have other capabilities that can accommodate these needs. (In fact, the sitemap is beginning to resemble a full-fledged program itself.) As a result, it can sometimes be a bit challenging for developers to make the sitemap do what they need, and its simplicity and readability can be sacrificed somewhat. Cocoon's developers are aware of this problem, however, and working to address it. They are now deciding how best to redesign the sitemap to handle these issues, and various ideas about the redesign have been discussed on the Cocoon Developers mailing list in recent months. As with all open source software, ideas from the development community are invited.

For More Info

We've only scratched the surface of the Cocoon technology here. I've omitted a great deal of material for brevity's sake. There's *much* more to Cocoon, and a detailed discussion could easily fill a book. (Indeed, the *Cocoon Developer's Handbook*, by Sue Spielman, is due out this year.)

If you're intrigued by what you've read here, I'd encourage you to start using Cocoon. There's no better way to learn than by hands-on development. The best way to approach Cocoon is to start with a small, simple site and build it up incrementally from there, learning as you go.

There's also loads of additional documentation available about Cocoon online, in fact, probably too much for a novice user. Again, I'd encourage you to approach it incrementally. Read a little at a time, learning more about each of the various components and techniques as you need them.

Resources

At the Cocoon Web Site (http://xml.apache.org/cocoon/):
- *Understanding Apache Cocoon:* http://xml.apache.org/cocoon/userdocs/concepts/
- *Cocoon 2 Idiots Guide:* http://xml.apache.org/cocoon/ctwig/
- *How to develop Web applications with Cocoon:* http://xml.apache.org/cocoon/tutorial.html
- *Cocoon Users mailing list:* cocoon-users-subscribe @xml.apache.org. (Details about acceptable content for the list at http://xml.apache.org/cocoon/mail-lists.html)
- *Cocoon Users mailing list archives:* http://xml.apache.org/cocoon/mail-archives.html

XSP Tutorials:
- www.suranyami.com/XSPtutorial/
- Spielman, S. (June 2002). *Cocoon Developer's Handbook*. Sams: www.amazon.com/exec/obidos/ASIN/0672322579/

darose@dti.net

Altova

www.altova.com

Listing 1

```
<?xml version="1.0"?>
<xsl:stylesheet
 version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
>
 <xsl:template match="/">
  <html>
   <head>
    <title>A Message From Cocoon</title>
   </head>
   <body>
    <xsl:apply-templates/>
   </body>
  </html>
 </xsl:template>

 <xsl:template match="message">
  <h1><xsl:value-of select="@text"/></h1>
 </xsl:template>
</xsl:stylesheet>
```

Listing 2

```
<map:components>
 <map:generators default="file">
  <map:generator
   name="file"
   src="org.apache.cocoon.generation.FileGenerator"
   label="content"
  />
 </map:generators>
 <map:transformers default="xsl">
  <map:transformer
   name="xsl"
  src="org.apache.cocoon.transformation.TraxTransformer"
  />
 </map:transformers>
 <map:serializers default="html">
  <map:serializer
   name="html"
   mime-type="text/html"
  src="org.apache.cocoon.serialization.HTMLSerializer"
  />
 </map:serializers>
 <map:matchers default="wildcard">
  <map:matcher
   name="wildcard"
  src="org.apache.cocoon.matching.WildcardURIMatcher"
  />
 </map:matchers>
</map:components>
```

Listing 3

```
<?xml version="1.0"?>
<map:sitemap
 xmlns:map="http://apache.org/cocoon/sitemap/1.0"
>
 <map:components>
  <map:generators default="file">
   <map:generator
    name="file"
   src="org.apache.cocoon.generation.FileGenerator"
    label="content"
   />
  </map:generators>
  <map:transformers default="xsl">
   <map:transformer
    name="xsl"
  src="org.apache.cocoon.transformation.TraxTransformer"
   />
  </map:transformers>
  <map:serializers default="html">
   <map:serializer
    name="html"
    mime-type="text/html"
  src="org.apache.cocoon.serialization.HTMLSerializer"
   />
  </map:serializers>
  <map:matchers default="wildcard">
   <map:matcher
    name="wildcard"
    src="org.apache.cocoon.matching.WildcardURIMatcher"
   />
  </map:matchers>
 </map:components>
 <map:pipelines>
  <map:pipeline>
   <map:match pattern="HelloWorld.html">
    <map:generate src="HelloWorld.xml"/>
    <map:transform src="Style.xsl"/>
    <map:serialize/>
   </map:match>
  </map:pipeline>
 </map:pipelines>
</map:sitemap>
```

Listing 4

```
<?xml version="1.0"?>
<page id="gu">
 <section>
  <row><field id="dsno_display"/></row>
 </section>
 <section>
  <row><field id="gdianame"/></row>
  <row><field id="gdianadres1"/></row>
  <row><field id="gdianadres2"/></row>
  <row><field id="gdiandate"/></row>
  <row><field id="gdianfee"/></row>
 </section>
</page>
```

Download the Code!
www.JavaDevelopersJournal.com

Capella University

www.capellauniversity.edu

Quintessence Systems Limited

www.in2j.com

Keith Brown J2SE Editor

Stress-Free Java

This month I'm at peace with the world so this editorial may seem somewhat relaxed and, dare I say, floppy. I've recently taken up Yoga – calmness in the mind, strength in the body, peace in the soul (or words to that effect). I mention this because I've felt the need to relax more outside the office, and this seemed the perfect way.

I know this is common in certain jobs where perhaps a lot of money is at stake with every minor decision. The kinds of jobs that frankly provide way too much stress and people problems.

I never expected stress as a Java developer!

Strange, is it not, that in this day and age, when most of us have satisfied the first four human needs of Maslow's Hierarchy by the time we're of working age, that we still feel stressed and require outside influences to relax us.

As a developer, where does the stress come from?

Last year I was involved in a project that required a sales reporting tool for an accounts department. The beginning and end of their financial year was the first Monday in April – so what would happen to the system when the year rollover came along?

Despite testing and testing until the cows came home, there's always just a little bit of doubt that creeps into your head and perhaps a sleepless night on the first Sunday in April, anticipating the dreaded phone call the next morning saying the system has gone pear-shaped. Not quite as stress-free as coding a Space Invaders applet.

Oh well! As Dennis Leary would say, "Join the club" (with a few more expletives I grant you). At least I'm not a 3D graphics artist on the next *Star Wars* film. (I've heard from certain sources that the approach on set is, "Oh, we'll just put that in later with special effects" – the 3D guys' eyes widening in disbelief and panic as they realize they'll have to animate 2,000 reptile-like creatures by next Tuesday.)

However, as I said, I am now at peace with the world, so let's move on.

Should I have been worrying? That's the question. I guess not. The start of the year came and my software trundled on with no problems. What would have happened had it gone wrong? Well, fortunately, because I was using Java, I didn't need to worry about the blue screen of death being presented to the user. Java's error-handling is a joy to behold, and the fact that it doesn't take down the whole system has gone a long way to cement Java as a serious 24/7 language of choice.

Recently our development team has been approaching the final stages of quite an extensive piece of software. To analyze and hopefully improve performance, we engaged the use of a profiler and, by jove, it worked a treat! There it was as clear as day! Line *x* to Line *y* took *way* too long. Sure enough, with a few tweaks the performance improved significantly.

Profilers are something I mentioned in a previous editorial, with reference to what the latest vogue was at JavaOne. They're something that every developer should have in his or her arsenal. It's really down to personal preferences as to what stage in your development you begin using a profiler. Some argue it should be from day one, while others use it more toward the end of a project. The first step is to become comfortable with the benefits a profiler can bring. The first time you use one, you're presented with a plethora of information that can take some time to digest.

But once you come to grips with it, you'll wonder what you ever did without one.

keith.brown@sys-con.com

Author Bio

Keith Brown has been involved with Java for many years. When he's not coding up client solutions for a European Java company, he can be found lurking in the corridors of conferences all around the world.

J2SE INDEX

New Atlanta Communications

www.newatlanta.com

Broken Windows in the Java World

The Java bug's evil twin

WRITTEN BY
Joe Xu

Not long ago I went with a couple of friends to a bar in lower Manhattan. While we were sipping Coronas, Jerry, our system architect friend, told us he had just inherited a high-profile J2EE system, along with one of the top Java teams in his company. "Now we know who's buying the beers tonight," we cheered.

Instead of a round of beers, Jerry decided to surprise us: "My first J2EE project might very well be my last." Apparently, Jerry had overestimated the flexibility of the project's code base, and for a guy obsessed with time lines, it cost him dearly – he missed his first deadline by weeks. What bothered him more was the fact that the team seemed to be giving up. "There are bandages everywhere," one developer told him before taking off. "But I thought the team was very good," I said. "They still are, but they just can't seem to keep up with the changes. The next thing you know, broken structures have mushroomed everywhere.…And now I'm working late every day."

As we were trying to raise Jerry's spirits, Ben, our project manager friend, pointed to a nearby TV and said cheerfully, "He can help you." It was a news clip about *Time*'s man of the year – Mayor Rudolph Giuliani.

Mayor Giuliani and Broken Windows

Turns out that Ben wasn't joking. If there is one tip Mayor Giuliani can share with Java developers, it's the power and simplicity of the "Broken Windows" theory. In a speech he gave at Lincoln Center in 1998, Giuliani pointed out that New York City had turned around, with overall crime dropping 44% over the last four years, and the "Broken Windows" theory, "an integral part of our law enforcement strategy," made this possible.

First written about in 1982 by political science professor James Q. Wilson and criminologist George Kelling, the "Broken Windows" theory states that "If a factory or office window is broken, passersby observing it will conclude that no one cares or no one is in charge. In time, a few will begin throwing rocks to break more windows. Soon all the windows will be broken, and now passersby will think that not only is no one in charge of the building, no one is in charge of the street onto which it faces. So more and more citizens will abandon the street to those they assume prowl it. Small disorders lead to larger ones, and perhaps even to crime." The net result of this strategy, Giuliani observed, was a transformation "that's improving the lives of millions of New Yorkers."

The State of Many Java Neighborhoods

Broken windows exist not only in the real world (possibly in a neighborhood near you), but also in the Java world, taking on various forms under the cover of functional code. Some broken Java windows are easy to spot and are better known in the industry as spaghetti code: massive nested if-else, code duplications across modules, and dead code, to name a few. On more than one occasion I've run across component instantiation routines with over 200 if-else, which can easily be replaced with one line of Class.forName().

Other broken Java windows are more subtle and require some digging. One such example is premature abstractions. Good abstractions make Java systems more flexible, but sometimes developers fall into the trap of trying to solve abstract problems before they even understand the concrete ones. The net result is unnecessary complexity that gets in the way later down the development stream (if you think no abstraction is bad, try bad abstraction).

Some broken Java windows are just plain inefficient. They work, but there are better ways of doing them (e.g., an optimized sorting algorithm implementing a comparable interface). Others are like time bombs that could blow away your whole system at any time; a security loophole comes to mind. When triggered, these sort of broken windows tend to be very costly, as we've seen in the past with eBay, whose 22-hour server downtime cost more than $5 million in return auction fees.

Contrary to popular belief, broken Java windows are not always the result of a manager or developer not knowing Java technologies any better, not drinking enough Java to do the job. Some windows are broken by inexperienced programmers who lack Java skills or by seasoned Java programmers who lack time and political support. A conversation like, "Hey bud. Hard wire Favorite Customer Inc. in the code. We'll all be out of here if we don't do this for them by next Monday. BTW, the big gun was just asking about this earlier," is not uncommon.

Some windows are broken the minute they're installed (early in the development stage), while others "creep in" later when dealing with high-speed, high-frequency business changes. The success that most Java systems enjoy has to take some blame for some of these changes. As a successful system becomes more popular, it attracts more users, who in turn demand more features and more flexibility quickly. The once ever-changing Java specifications are a good case in point. Also, most of the Java systems find their homes in Internet-driven business. Unfortunate-

Ashnasoft Corporation

www.ashnasoft.com

> Broken Java windows are given a license to do whatever they like for as long as they like

ly, the Internet industry is constantly reinventing itself with businesses trying different models. Some models deal with how you charge, some with how you deliver the services. Some work, some don't. The dynamic nature of this environment can break all windows if not managed carefully.

What's the Big Deal About a Few Broken Java Windows?

Despite their diverse appearances, all broken Java windows seem to be sticky – they have a tendency to stay around for a long time, sometimes even throughout the whole system's life cycle. Although developers are not particularly fond of them, they don't lose sleep over them either. After all, broken windows do work with respect to specified behaviors, and what's a better alternative to not delivering the system on time?

This also explains the fact that developers generally don't introduce bugs intentionally, but the same thing can't be said about broken windows. People on the business side don't seem to mind them either – they might not even be aware of their existence. And if they do, a typical response is along the lines of "Does it work?" "Yes, but it's a bad thing to do." "Well, if it ain't broke, don't fix it. We are way overloaded already." So broken Java windows are given a license to do whatever they like for as long as they like.

Another common characteristic is that broken Java windows "replicate" fast. Like worms, they spread swiftly, leaping from module to module throughout the whole system. But how? If an object or routine is broken, developers observing it will conclude that no one cares or is in charge. In time, a few will conclude further that the system is not worth the effort (i.e., "All the rest of this code is crap") and more routines will get broken. Soon all the routines will be broken, and now developers will think that, not only is no one in charge of the object, no one is in charge of the subsystem to which it belongs. So more and more broken windows will pop up all over the subsystem and its neighbor subsystems.

Small disorders lead to larger ones, and perhaps even to project cancellation and firings. In other words, broken Java windows create a delusion of dysfunctional systems, even though a system might be perfectly fine up to that point. Once developers are convinced that the current system is crap, it becomes a license for them to break some more Java windows. In contrast, if developers are convinced otherwise, they would think twice about breaking any window – after all, we Java developers are no different than any other artist – we love and appreciate beautiful things. With that appreciation, we do everything we can to preserve them. It's these two characteristics that enable broken Java windows to accelerate their damages.

The Little-Known Evil Twin of the Java Bug

If a broken window sounds just like another name for a Java bug, it's not – it's the more evil twin of a Java bug that we don't usually hear about. A bug is usually defined as something that's not functional and is visible to the business. On the other hand, a broken window is functional and often invisible, but has a mid- to long-term negative impact on the health of the system and the development team.

In many ways comparing a Java bug to a broken window is like comparing a cold to an early-stage terminal disease – without warning signs it's harder to detect (remember why your doctor recommends regular checkups), but in the end it proves far more lethal than a cold. Like any terminal disease, broken windows slowly deteriorate system internals over time, until one day the system is completely drained. In other words, it becomes so "bloated" or badly structured that you'll find yourself spending an enormous amount of time and effort to implement a simple change. At the same time, it "deteriorates" the most precious resource of all – the development team. It bumps up the turnover rate so high that no one really knows about the system anymore.

As Giuliani pointed out, if a climate of disorder is allowed to take root, more serious antisocial behavior will increase. "There's a continuum of disorder." Obviously, bugs and broken windows are two different "disorders." But they are part of the same continuum, and a climate that tolerates one is more likely to tolerate the other. "…A city in which an increasing number of people respect and are willing to accommodate the rights of others is a city that's moving in a progressive direction." By the same token, a Java system in which broken windows are "arrested" is a system that's moving in a progressive direction with true Java spirit.

Does that mean that as Java developers, our new mission is to "hunt down" all broken windows mercilessly ASAP? In a time-starved society, that's often impossible. What we need instead is a different approach, an approach that allows us to make steady progress toward our real goal – delivering functionality and, at the same time, fixing broken Java windows, an approach that says a Java team should only fix windows at natural project breaks, and prioritize their broken windows before trying to fix them.

Identify Broken Java Windows

Some Java developers react to broken windows by installing a "broken Java window detector." Since broken windows are a legitimate concern throughout development, no new functionality can be considered done until it has passed the detector. This practice might sound good and complete on paper, but in real life it is the recipe for killing a good Java system. The first symptom is the fast-dropping development velocity. And instead of listening to the users' needs, concentrating on business scenarios, and getting functionality done right, every thought is now interrupted by the question "Is this a broken window?" A team will most likely miss the chance to make a big leap in their development progress.

Plan on identifying broken windows toward natural breaks, such as when a module is expected to be complete, or before every minor release. Allocate time in your project plan to do so. It helps keep your team and project stakeholders informed of your priorities so they'll act accordingly.

Categorize Broken Java Windows

Once identified, you could "arrest" broken Java windows one at a time. In

Parasoft Corporation

www.parasoft.com/jdj6

my experience though, some Java windows are broken due to a particular design or approach that spans multiple areas. For instance, without a class factory, you might find component instantiation code scattered over multiple subsystems. In this case, if we arrest broken windows one at a time, we might have ended up with different fixes, as the windows might get assigned to different developers. In other words, we miss the benefit of finding commonality. Another approach is to profile all identified broken windows, take into account several modules in which the same type of broken window occurs, and you may find yourself a different fix.

Profile your broken windows, categorize them, and you just might be able to fix two or more windows at one time.

Prioritize Broken Java Windows

Once all broken windows are categorized, our gut reaction is to "arrest" them all. Unfortunately, due to time constraints, that would force you to either commit to less functionality than you should be implementing or fail to deliver the system on schedule, in which case no one would care about broken windows anyway.

Prioritize your broken windows before putting a lot of energy into fixing them. Now is also a good time to involve project stakeholders. They might dislike certain broken windows more than others. On one project, just when I thought everyone would care less about executable size than memory footprint, some stakeholders were more than happy to indicate otherwise. And of course they were right.

Arrest Broken Java Windows

Once prioritized, we know we'll be arresting the ones that offer the biggest bang for the buck. There are many ways to actually fix broken Java windows. The best-known technique is probably refactoring and JUnit.

If we can't fix all identified broken windows, what do we do with the rest of them? Keep them in your ongoing broken windows list. More important, do the little things that show the team cares. My personal favorite is to write down a little comment next to the code starting with the word "FIXME" for bad functionality or "TODO" for a missing one. That would buy the team some time psychologically, until the next time slot for broken Java windows comes around.

Conclusion

If we're investing time and money in Java bugs, shouldn't we do the same for their evil twin? If broken windows in your Java world are not getting fixed, maybe it's time to seriously think about moving to a new Java neighborhood. After all, who likes to have a time bomb in their Java backyard?

Author Bio
Joe (Leizhou) Xu, an AVP with Credit Suisse First Boston, delivers large-scale J2EE systems for Fortune 1000 and e50 companies. He holds an MS and a BS in computer science.

References

- Giuliani, Rudolph W. "The Next Phase of Quality of Life: Creating a More Civil City": www.nyc.gov/html/rwg/html/98a/quality.html
- Wilson, T. (1999). "The Cost Of Downtime." *InternetWeek*. July. www.internetweek.com/lead/lead073099.htm
- Wilson, J.Q., and Kelling, G.L. (1982). "Broken Windows: The police and neighborhood safety." *The Atlantic Monthly*. March. www.theatlantic.com/politics/crime/windows.htm
- Hunt, A., and Thomas, D. (1999). *The Pragmatic Programmer: From Journey man to Master*. Addison-Wesley.
- Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley.

joe.xu@bigfoot.com

Northwoods Software

www.nwoods.com/go/

aqriaus

p/u

Dice

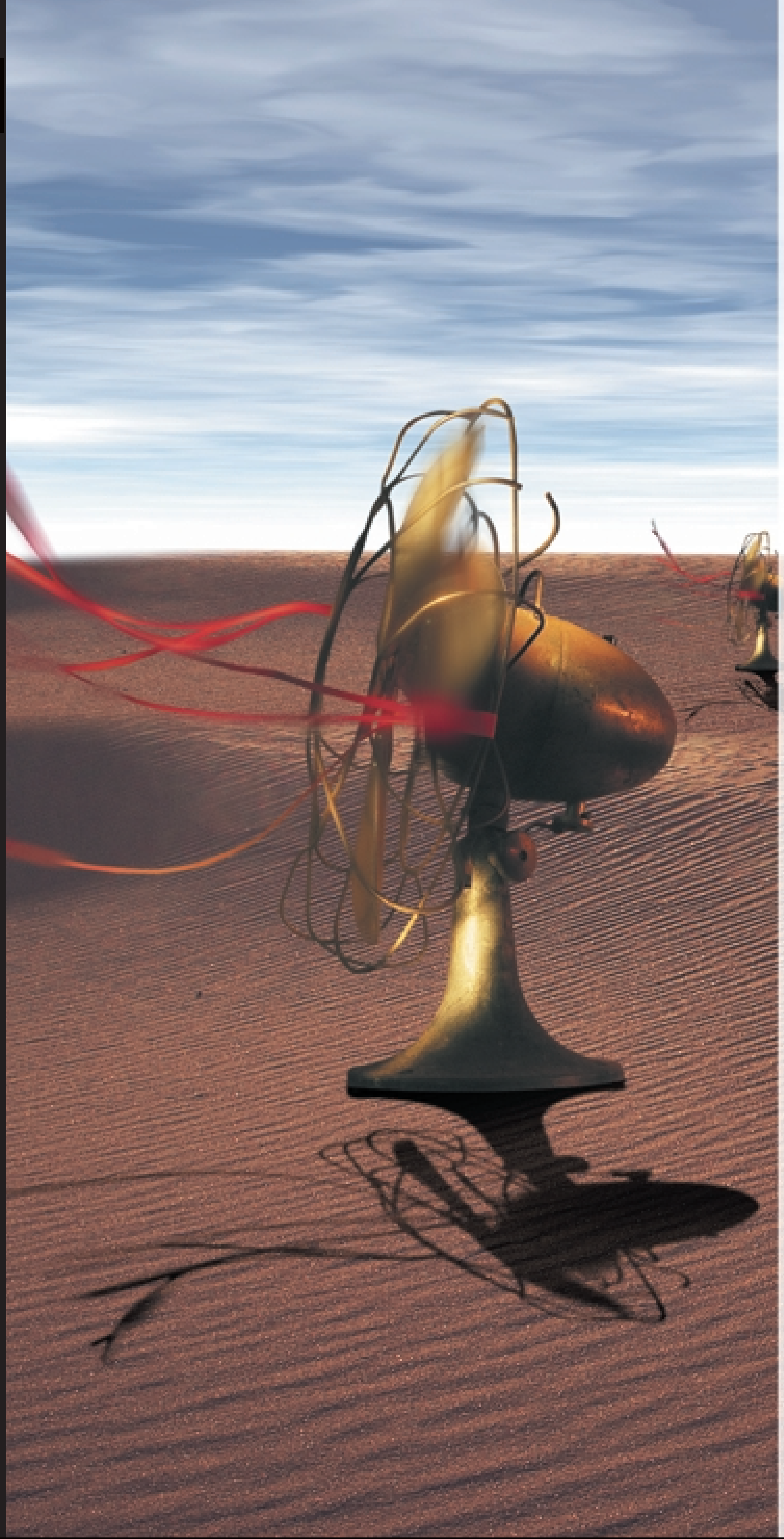
www.dice.com

Create a reusable class that allows access to the AIM network

Interfacing to AIM with Java

written by Jeff Heaton

Instant messaging has become very popular in recent years, earning it a deserved spot with the "killer apps" – browsing and e-mail. Most of the Internet's killer apps have spawned a host of accessories to be used in conjunction with the app. Instant messaging is just starting this process.



Many clients are appearing that can often send messages over various protocols. But do other models exist, besides simply sending instant messages between human users?

Several companies are already putting computers on the receiving end of instant messaging. One such company, ActiveBuddy (www.smarterchild.com), put several "screen names" online that allow users to ask "virtual buddies" about such things as movies, weather, and stock quotes. Programs such as this require intimate knowledge of how the instant messaging system works.

This article shows you how to communicate with America Online's (AOL) Instant Messenger, known as AIM, by creating a reusable class that allows access to the AIM network, which you can use to create an instant messaging application.

AOL Protocols

There are many instant message clients that can talk to the AOL network, for example, programs such as Trillian, Jabber, and GAIM. In the past few months these programs have come under fire from America Online as AOL tries to keep unauthorized programs off their network. Why does AOL try to stop some clients and allow others to continue? The answer has to do with what protocol the instant messaging client uses.

When America Online first allowed Internet users to access the AIM network, they had to provide a protocol that would allow the Internet version of AIM to communicate back to the AOL network. This protocol is called OSCAR – Open System for Communication in Realtime – a misnomer. OSCAR is not an open system, or at least not "open" in the sense of GNU-style licensing.

OSCAR is a closed protocol, as AOL has never published its specification. Any large-scale use of the OSCAR protocol has always been vigilantly blocked, but this has not stopped programmers. A great deal of information is available on the Internet about the OSCAR protocol and many third-party instant message clients are based on it. I wrote an OSCAR client in Java that communicates with AOL using this protocol. However, if this client runs longer than 5–10 minutes, AOL sends me the following instant message just before their network disconnects my client.

You have been disconnected from the AOL Instant Message Service for accessing the AOL network using unauthorized software. You can download a FREE, fully featured, and authorized client here, http://www.aim.com/index.adp.

This article is not about how to use the OSCAR protocol. If you're interested in OSCAR, you can download my client from www.jeffheaton.com/im. If you use the OSCAR protocol for a large-scale project, you'll be playing "cat and mouse" with AOL. AOL's actions indicate that they do not want third parties using it. Products such as Trillian use the OSCAR protocol, and, as a result, users must frequently upgrade to a new version of Trillian that thwarts AOL's latest attempt to block them.

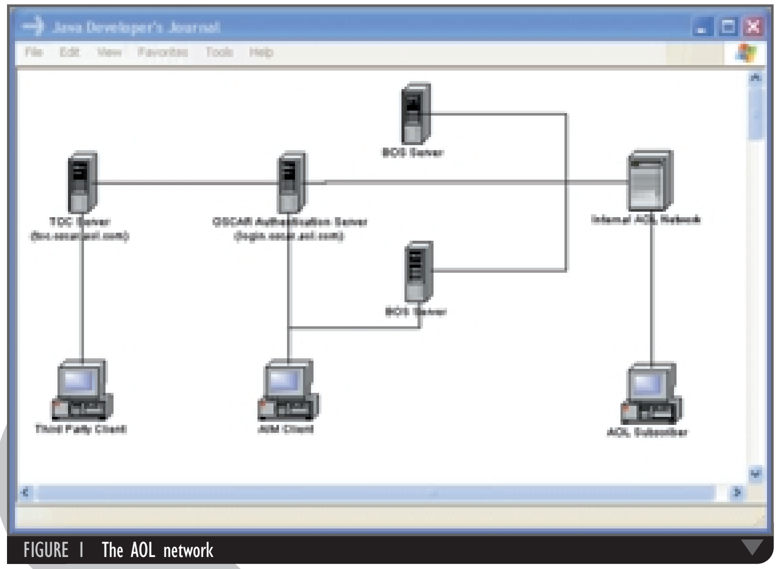
This article focuses on another AOL protocol named TOC (Talk to Oscar). TOC was created by America Online to allow open access to their instant message network; it was created prior to their merger with Time Warner. One of the conditions for this merger, set out by the U.S. government, was for AOL to open its instant messaging protocol. So AOL in essence created a wrapper. The TOC protocol sits on top of the OSCAR protocol and allows third-party applications to access the AIM network. AOL also released an open source, instant message client that's capable of using the TOC. The structure of the AOL network is shown in Figure 1.

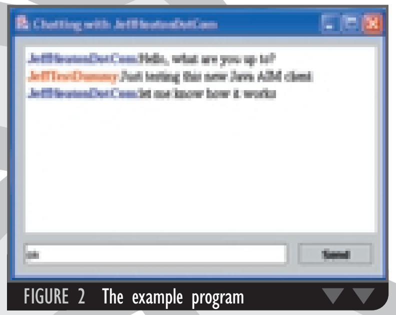
As you can see from this figure, AOL subscribers connect directly to the AOL internal network. The AOL AIM clients go directly to the authentication and BOS (used for load balancing) servers, and the TOC clients connect through the TOC server. This results in three connection classes that can all communicate with each other.

The open source AIM client released by AOL was written in Tcl, and was called TiC (TOC is Cool). AOL no longer officially supports the TiC client, so it can't be downloaded from the AOL site. TiC is now maintained as an open source client on Source Forge (www.sourceforge.com).

Clients that use the TOC protocol do not face the same blocks that OSCAR clients face. Unfortunately, there aren't any popular clients available for the Windows platform that make use of the TOC protocol. Trillian, for example, uses only the OSCAR protocol. Perhaps the most common client that supports the TOC protocol is GAIM. Unfortunately, it's written in C and supports only the UNIX platform. There's not much open Java source code for using the TOC protocol, which is why I wrote JavaTOC as an example for this article. Using the JavaTOC class, you can communicate with both the TOC protocol and the AIM network. This article presents a sample AIM-compatible client (see Figure 2), and the complete source code is available on the ***JDJ*** Web site, www.sys-con.com/java/sourcec.cfm.

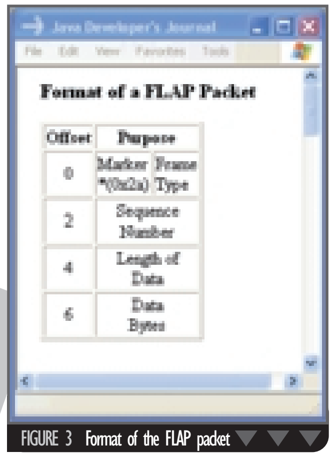
If TOC is the protocol that AOL wants developers to use, why do most third-party programs use OSCAR? There are several reasons for this. First, OSCAR was available long before TOC. Many third-party programs were already written when TOC was released. Second, AOL has not been that good about adding all the latest features to TOC. To be able to send files and access the advanced "broadband" streaming content that AIM now contains, you must use the OSCAR protocol, since such features have not been added to TOC. However, for basic instant messaging and chat, TOC has all the necessary features.


Java Developer's Journal
File Edit View Favorites Tools Help
BOS Server
TOC Server (toc.oscar.aol.com)
OSCAR Authentication Server (login.oscar.aol.com)
Internal AOL Network
BOS Server
Third Party Client
AIM Client
AOL Subscriber
FIGURE 1 The AOL network



JeffHeatonDotCom:Hello, what are you up to?
JeffTestDummy:Just testing this new Java AIM client
JeffHeatonDotCom:let me know how it works

FIGURE 2 The example program


Java Developer's Journal
File Edit View Favorites Tools
Format of a FLAP Packet

Offset	Purpose	
0	Marker *(0x2a)	Frame Type
2	Sequence Number	
4	Length of Data	
6	Data Bytes	

FIGURE 3 Format of the FLAP packet

Logging into TOC

The complete source code for my implementation of the TOC protocol can be found in the files "JavaTOC.java" and "Chatable.java". These two files can easily be added to any project that requires the use of instant messaging. To see how the JavaTOC class was constructed, you must first see how the TOC protocol works.

To log in to AIM you must have a screen name and password. If you don't have an AIM screen name, you can easily obtain one for free from http://aim.aol.com. The JavaTOC class defines a method called "login" that accepts a screen name and password. When you call login, it begins the login process.

To log in to TOC you're required to send and receive data using FLAP, a special packet type. (It's unknown what FLAP stands for, as AOL has always used this name.) FLAP also forms the foundation of the OSCAR protocol. A FLAP packet consists of a header of six bytes followed by a variable length data packet (see Figure 3). The first byte of a FLAP packet is always the asterisk (*), which is hex code 0x2b. Following the identity byte is the frame type. TOC uses two frame types. One frame type indicates a "sign-on" packet, and the other a normal "data" packet. Most packets are data packets.

Following the first two bytes is a word representing the sequence number. Why AOL needed a sequence number on a protocol that uses TCP/IP is unknown. However, you must always provide an ever-increasing sequence number or your client will be abruptly disconnected. Sequence numbers can begin with any number and must always increase by one. The sequence numbers used by the client are completely separate from the sequence numbers sent to you by AOL. Following the sequence number is a word that specifies the length of the data section only and does not take into account the size of the FLAP header.

Both the length and sequence numbers are stored as little endian numbers, which means the last byte is the least significant. For example, the value 1024 would be stored as the byte 0x04 followed by 0x00, not the other way around. For more information on little and big endian numbers, see my article "A Class for Reading Binary Files" (*JDJ*, Vol. 6, issue 7). To allow these words to be written correctly, JavaTOC contains a method called "writeWord" that will write a word to an "InputStream" in a format that's correct for TOC.

```
protected void writeWord(short word)
  throws IOException
{
  os.write((byte) ((word >> 8) & 0xff) );
  os.write( (byte) (word & 0xff) );
}
```

The JavaTOC class also contains a useful method for writing FLAP packets to a socket. This handles the sequence number and lengths for you. Simply pass the string the writeFLAP method you'd like transmitted and the frame type (signon or data).

LOOX Software Inc.

www.loox.com

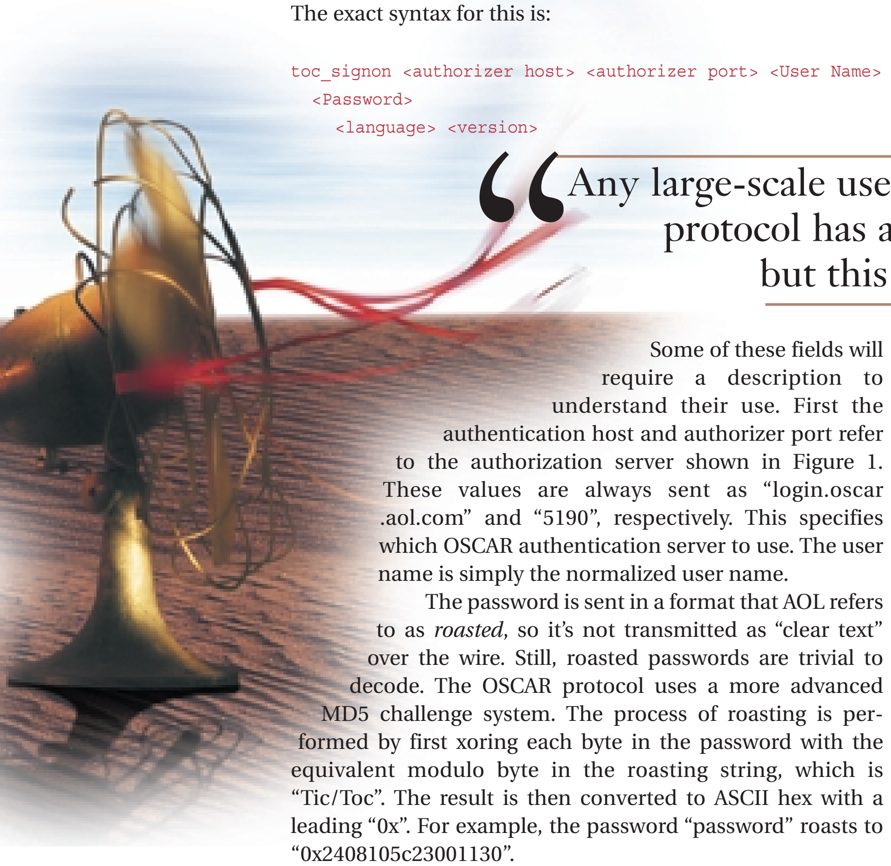
```
protected void sendFlap(int type,String str)
  throws IOException
{
  int length = str.length()+1;
  sequence++;
  os.write((byte)'*');
  os.write((byte)type);
  writeWord(sequence);
  WriteWord(length);
  os.write(str.getBytes());
  os.write(0);
  os.flush();
}
```

All the TOC commands are strings. The only part of the TOC protocol that's binary is the FLAP packets that enclose the TOC commands. Now, with the FLAP packet defined, we can log into TOC. The first step is to open a socket to the TOC server. The host name for the server is "toc.oscar.aol.com", and the port is 9898. Once a socket connection has been opened, the string "FLAPON\r\n\r\n" must be sent to the TOC server. Upon receiving this, the TOC server will reply with a FLAP pack containing the data bytes 0x00, 0x00, 0x00, and 0x01. This is the current version of FLAP being used, version one.

Next a FLAP signon packet must be sent. This is a normal FLAP packet with a data section that contains the following: first, the version numbers 0x00, 0x00, 0x00, and 0x01, then a word of 0x0001. This indicates that a user name will follow. The next word represents the length of the screen name. Following the length is the normalized representation of the screen name – all spaces are removed and the entire name is converted to lowercase.

Once the FLAP signon packet has been sent, the first TOC command must be sent. This is the "toc_signon" command. The exact syntax for this is:

```
toc_signon <authorizer host> <authorizer port> <User Name>
  <Password>
    <language> <version>
```



Some of these fields will require a description to understand their use. First the authentication host and authorizer port refer to the authorization server shown in Figure 1. These values are always sent as "login.oscar .aol.com" and "5190", respectively. This specifies which OSCAR authentication server to use. The user name is simply the normalized user name.

The password is sent in a format that AOL refers to as *roasted*, so it's not transmitted as "clear text" over the wire. Still, roasted passwords are trivial to decode. The OSCAR protocol uses a more advanced MD5 challenge system. The process of roasting is performed by first xoring each byte in the password with the equivalent modulo byte in the roasting string, which is "Tic/Toc". The result is then converted to ASCII hex with a leading "0x". For example, the password "password" roasts to "0x2408105c23001130".

"Any large-scale use of the OSCAR protocol has always been vigilantly blocked, but this has not stopped programmers"

TOC currently supports only English; as a result "english" is the only value that should ever be sent as a language. The version field specifies which client is logging in and can be any value. If the login is successful, TOC server will respond with "SIGN_ON:version". If a failure happens, an "ERROR:error number" response is returned.

At this point the user is not signed in yet. To complete the login process the user's client must send the "toc_init_done" within 30 seconds of beginning the login process. Before the login method sends the "toc_init_done" command, a few configuration items are taken care of. First the "toc_add_buddy" command is sent to specify the user who's logging in as a buddy. Then the "toc_init_done" command is sent to complete the signon process.

After signon, a few configuration settings must be sent. The "toc_set_caps 09461343-4C7F-11D1-8222-444553540000 09461348-4C7F-11D1-8222-444553540000" command is sent. The two GUIDs specify the ability to send messages and buddy support. Finally, the permit and deny lists are cleared by sending the "toc_add_permit" and "toc_add_deny". These are lists that allow you to include or exclude certain people. With this complete, the client may now send and receive instant messages.

Logging out of TOC is easy. Simply breaking the socket connection by using the "close" method of the "Socket" class is sufficient.

Sending and Receiving Instant Messages

Sending and receiving instant messages with TOC is fairly easy. To send, the send method of the JavaTOC class should be called:

```
public void send(String to,String msg)
{
  try {
    this.sendFlap(DATA,"toc_send_im " + normalize(to) + "
     \"" + encode(msg) + "\"");
  }
  catch(java.io.IOException e){
  }
}
```

This method accepts two parameters that specify who the message is going to and what the actual message is. The target's screen name is normalized, as previously defined. The message is encoded to fit inside quotes. Encoding the message involves setting any carriage returns to HTML <br> commands and putting a \ in front of characters such as quotes. AIM messages may contain HTML code.

Receiving an instant message is accomplished by the event loop contained in "JavaTOC". Because events can be received at any time from the TOC server, the program must be ready to handle them. The JavaTOC class contains a method, processTOCEvents, to do this:

```
public void processTOCEvents()
  throws IOException
{
  for(;;)
  {
    String str = this.getFlap();
```

InstallShield Software Corp.

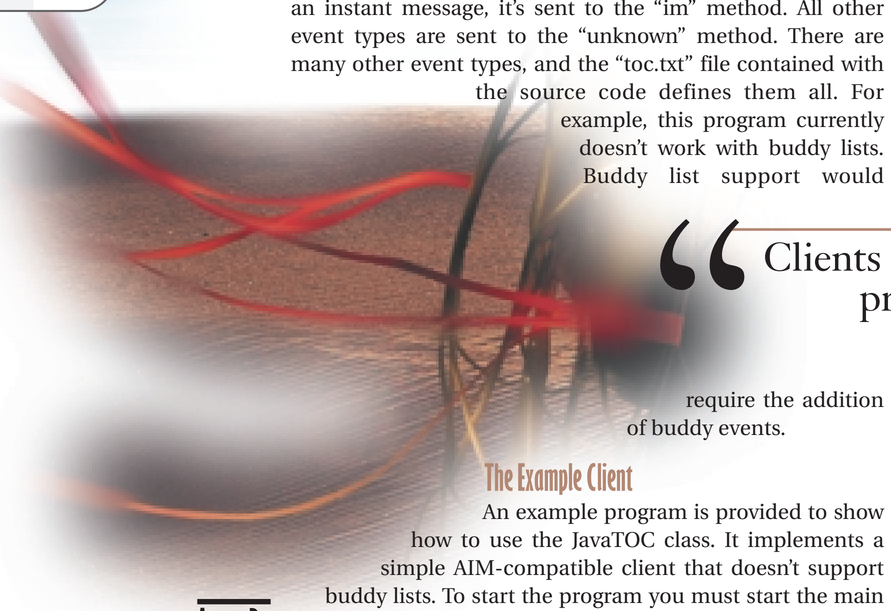
www.installshield.com

```
      if(str==null)
        continue;
      if(str.toUpperCase().startsWith("IM_IN:")) {
        handleIM(str);
      }
      else if(str.toUpperCase().startsWith("ERROR:")){
        handleError(str);
      }
      else{
        owner.unknown(str);
      }
    }
  }
```

As you can see, this code is an endless loop. This method is intended to be run as a background thread. This is the approach taken by the example client presented with this article. As instant messages come in through the TOC "IM_IN:" command, they're routed back to another class that's waiting for them. This class must implement the "Chatable" interface. These interfaces specify the TOC events that will be sent to a class that's using the JavaTOC class. This interface is shown here.

```
interface Chatable
{
  public void unknown(String str);
  public void error(String str,String var);
  public void im(String from,String message);
}
```

As messages are received, they're routed to the "error" method if the event is a notification of an error. If the event is an instant message, it's sent to the "im" method. All other event types are sent to the "unknown" method. There are many other event types, and the "toc.txt" file contained with the source code defines them all. For example, this program currently doesn't work with buddy lists. Buddy list support would require the addition of buddy events.



> "Clients that use the TOC protocol do not face the same blocks that OSCAR clients face"

The Example Client

An example program is provided to show how to use the JavaTOC class. It implements a simple AIM-compatible client that doesn't support buddy lists. To start the program you must start the main class "ChatBuddies" using the command "java ChatBuddies". A JAR file named "JavaTOC.jar" is included and can be executed in order to see the example. Some computer systems allow you to simply double-click this file to execute the example.

The example program (shown in Figure 2) is implemented using three Swing "JFrame"-derived objects. The main frame used is the "ChatBuddies" class. This class would normally be where a buddy list would be. Instead, it displays an input field that allows you to specify the screen name of the person you're sending the message to. Clicking OK will launch the "ChatWindow" frame, which is the typical instant messenger–type chat window. If you're talking to more than one person at a time, multiple windows are displayed.

The "ChatBuddies" class contains the background thread that processes all TOC events by calling the "processTOCEvents" method of the "JavaTOC". The "run" method of this thread is shown here.

```
public void run()
{
  try {

if(toc.login(login.userID.getText(),login.password.getText
    ())){
    this.setVisible(true);
    this.setTitle("Logged in: " + login.userID.getText()
      );
    login.setVisible(false);
  }
  else
    return;
  toc.processTOCEvents();
  }
  catch(Exception e)
  {
  }
}
```

First the user is logged in. If the login fails, a TOC error event will occur and be displayed by the "processTOCEvents" method. This background thread handles all incoming messages.

Conclusion

A lot can be done with instant messaging technology. By using the Java code provided in this article, you can write instant messaging–enabled programs. This code implemented a client. It's also possible to write server type applications that wait for connections and provide data. An example of this would be a program that sends an instant message to indicate something failed on a server.

The example program provided here is not a complete instant message application. Adding such features as a buddy list, HTML, and saved configurations would greatly enhance its usability. Yet, despite these limitations, it shows the fundamentals of how to work with AOL's instant message architecture.

References

- *Glossary of terms used by Oscar/ToC given by AOL:* www.aim.aol.com/javadev/terminology.html
- *The GNU protocol specification by ToC as released by AOL:* www.jeffheaton.com/im/toc.txt
- *One of the more complete documents describing the internals of the OSCAR protocol:* http://aimdoc.sourceforge.net/OSCARdoc/section2.html
- *The TiC client:* http://sourceforge.net/projects/tik/
- *Ethernet packet analyzer:* www.ethereal.com
- *The AOL instant messenger (AIM):* http://aim.aol.com

Author Bio

Jeff Heaton, a software designer for the Reinsurance Group of America (RGA), is a member of the IEEE and a Sun-certified Java programmer. He's the author of Programming Spiders, Bots, and Aggregators in Java.

heatonj@heat-on.com

Actuate Corporation

www.actuate.com/info/jbjad.asp

Using the Java Native Interface Productively

Simplify repetitive tasks

WRITTEN BY ANDREW J. CHALK

Although we try to make our applications pure Java, outside forces sometimes make this impossible. We had such a case recently in our shop when we had to interface to an external device with an API that supported C language calls.

This is a typical case for the Java Native Interface (JNI). The JNI provides Java programs with a gateway to other languages and enables applications written in other languages to invoke the Java Virtual Machine.

This article focuses on the first of these two uses. Specifically, it discusses supporting a C/C++ API in Java to allow a Java application to use it. This is probably the situation in which the JNI is most frequently employed in production environments. Indeed, the seminal work on the JNI, *The Java Native Interface: Programmer's Guide and Specification* by Sheng Liang, devotes a chapter to this question. There's no theory involved, but if you use the following techniques you should save yourself many hours of work.

Phase one of the JNI programmer's acclimatization process: dump the sacred principle of Java development – platform independence. It must, as a purely logical implication, go out the window. All the C/C++ code in this article compiles under GNU C/C++ on Solaris and Microsoft Visual C++ on the Win32 platform. Other platforms would have to be tested individually.

I'll focus on the practical problem of implementing a large and complex API using the JNI within a production environment. Space constraints required Liang to discuss passing parameters that consisted of Java primitives from Java to C/C++ (hereafter C++, unless the distinction matters) and sending the C++ return code to the Java application. However, most APIs are more complex. This is something not frequently addressed in the literature, despite its prevalence in the real world. APIs are not restricted to primitives as parameters but pass structures and classes to the device. They not only use return statements but also assign values to the passed parameters for use by the calling program. They present questions for interpreting Java classes in C++, handling structure alignment and data format issues, and implementing code that is source compatible with both 32-bit and 64-bit processors.

AUTHOR BIO
Andrew J Chalk, is president of Magna Carta Software, Inc., in Plano, Texas.

The JNI, Power – at a Price

Every developer who has used the JNI is aware that it's powerful and comprehensive. However, the price of these features is an inordinate amount of work to accomplish some very mundane tasks. In particular, parameter handling is a time- and code-intensive business that's subject to errors and latent bugs. Consider the code in Listing 1, which might be described as the simplest function to access a JNI. The native function is declared at (1) in the listing. (In Listings 1, 4, and 6 I've numbered specific lines for reference purposes.)

The method passes one integer to a C++ native function. The native function prints the passed parameter to stdout and also returns it. The native code is shown in Listing 2. Note the obscure function name, which is generated by javah.

The native function just prints the integer parameter to stdout. How this appears depends on your Java development environment. In Borland JBuilder the output from the native code and the System.out.println()call both appear in the IDE message window, as shown below.

```
Value is 5
Value sent and returned is 5
```

The native code prints the first line and Java prints the second. The important points about this example are:

- The integer parameter is passed from Java to C++ as a primitive, so the native code can access it directly.
- The same rule applies to all other Java primitives.
- The use of the JNI is straightforward, and the native code can be developed rapidly as long as its use is confined to passing primitives.
- The native code may require modification for different platforms. While ISO standardization has made C++ source code fairly portable, we're out of the area of Java binary compatibility.

Most data types are more complicated than primitives. In Listings 3 and 4 the Java and native code for passing and returning a string type are shown. The following is the output from these listings.

```
Value is Hello World
Value returned is Goodbye World
```

While there's virtually no change in the Java code (essentially just a change of the operative variable type from int to string), the native code is substantially more involved. Because a string is neither a primitive type nor architecturally the same as any string in the native language, it must be handled differently. First, the JNI function GetStringUTFChars() [at (1) in Listing 4] both converts the string to the C++ single-byte character set and retrieves a pointer to the resultant object (other JNI functions can handle conversion to C++ wide, e.g., Unicode, strings). Second, the JNI function ReleaseStringUTF() frees the memory allocated for the converted string. Third, the JNI function NewStringUTF() allocates a Java string from a C++ single-byte character string to return a string to Java. The important point about Listing 3 is that passing strings increase the amount of native code necessary to process the parameter.

String parameters don't present even the normal level of variable complexity. Real-world data types are likely to be represented as classes in Java and structures or classes in C++. Listings 5 and 6 provide the Java and C++ code for passing a simple class.

The Java code in Listing 5 is very simple. It just instantiates instances of

HiT Software

www.hitsw.com/xmlrdb

ClassAccess and Device2 where Device2 aggregates an instance of Device1, which it instantiates in the constructor chain. The code is straightforward, indeed mechanical. However, the native method (see Listing 6) is completely different. While not conceptually sophisticated, it's a complicated tangle of repetitive operations on different data types.

First, at (1), the code fetches the class of the object passed to the native function. This is the Device2 object passed in the Java line classAccess.accessClass(dev2). It then proceeds to retrieve each parameter in turn. The block at (2) retrieves the int field intValue. The code at (3) retrieves the long field longValue. The block at (4) is more involved as it retrieves the string field stringValue and resembles Listing 3. Finally, the block that begins at (5) shows the more involved case of obtaining a field from an object contained within another object. In this example the Device2 object contains a Device1 object. It's the int member of Device1, intValue, that we wish to eventually obtain. [This is obtained at (6).]

Simplifying the Implementation

Code like this will be repeated unless we find a way to move its regularities into separate functions. The string and contained object cases generate the most overhead, so they're the ones chosen in Listing 7. Now the native code in Listing 5 can be abbreviated to that shown in Listing 8.

Furthermore, the two access functions repeat the simplification that they provide here in each native function they're called in. In each instance that we use this code we reduce the amount of code required to be implemented in the project by roughly two-thirds.

The JNI is a powerful, comprehensive solution

Assigning Data to Java Structures

Now we're at the point at which productive use of the JNI became a major issue in our shop. The vendor API that we supported required some structure passing and what we discussed earlier proved useful for this. However, the API's main purpose was to return data from the vendor's device, which it did by providing over 100 functions that required a pointer to an API structure to be passed to an API function. The vendor library allocated memory for the structure and instantiated the fields with the requested data. This worked well for the C++ programmer who could examine the data by dereferencing the pointer after the API function call returned.

The Java programmer acquired a new set of problems. We decided to define a Java class for each C++ structure (a practice recommended by Bloch). Each field of each API structure had to be assigned to a member of the corresponding Java class, passed as a parameter. The one-to-one mapping of Java classes to C++ structures gave us over 60 Java classes. The number of structure members meant that over 500 member assignments had to be made. Code resembling that in Listings 1–5 would have to be written and, more important, maintained – a daunting prospect.

We turned to a set of functions like those in Listing 7. However, now these functions would read a member variable from a C++ structure and assign the value to the Java class (see Listing 9 for an example). For continuity, we continue to use the Java classes Device1 and Device2 defined earlier. Their C++ equivalents are:

```
struct Device1
{
  int intValue;
};

struct Device3
{
  int intValue;
  long longValue;
  char * stringValue;
  Device1 dev1Value;
};
```

Assume that at runtime they contain the values shown below:

```
struct Device1 dev1 = {10};

struct Device3 dev3 = {
  5,
  8,
  "Hello World",
  {10}
};
```

Listing 9 is the Java program that will print out these values.

The C++ code (see Listing 10) shows the implementation of the native method assignClass. One interesting feature is that it shows how to make assignments to nested objects (Device1 is aggregated by Device3 in Listing 9). Just as in the case of passing parameters from Java to C++ we provide a set of routines to expedite the process and make the code more reliable (see Listing 10). Listing 11 can then be replaced with Listing 12. (Listings 11 and 12 can be downloaded from the *JDJ* Web site, www.sys-con.com/java/sourcec.cfm.)

Benefits

The family of Parse... functions has substantially reduced the amount of code required to make the necessary assignments to the Java object passed as jOutput and its contained objects. Furthermore, we've moved the intricate code in which errors can occur into a set of reusable functions. This expedites the implementation of a large vendor API in Java and makes maintenance more straightforward. These functions are extensible to other variable types as necessary. We've implemented only the ones that we need.

Summary

The JNI is a powerful, comprehensive solution to the problem of accessing an API written to support other languages, especially C or C++. That power comes at the price of intricate parameter manipulations. In this article we addressed the problem of using the JNI in a production situation and presented a set of techniques that simplify most of the repetitive tasks of passing information to the native code and getting it back into our Java program. These techniques reduced by around two-thirds the amount of C++ code that had to be written. These techniques can be extended to cover other constructs, and it's a pragmatic choice as to which should be implemented in a particular project.

Postscript

After completing the project in which these techniques were developed, another JNI project came up. Implementation of the native parts of the project took about a quarter of the time that we had forecast, based on raw coding of JNI function calls. The code also passed all unit tests on Solaris and Windows the first time, an indication of the reliability gains of a unified method of handling parameter passing. While an unscientific estimate (different APIs, etc.), the difference in terms of schedule improvements was very noticeable.

References

- Liang, S. (1999). *The Java Native Interface: Programmer's Guide and Specification*. Addison-Wesley.
- Bloch, J. (2001). *Effective Java Programming Language Guide*. Addison-Wesley.

achalk@magnacartasoftware.com

Sitraka

www.sitraka.com/jprobe/jdj

Listing 1: Passing an int to the JNI

```
package jni_article_;

public class IntegerAccess {
  static {
    System.loadLibrary("EasyJNI");
  }
  private native int accessInt(int value);     // (1)
  private int value;
  public IntegerAccess() {
    value = 5;
  }

  public static void main(String[] args) {
   IntegerAccess integerAccess1 = new IntegerAccess();
System.out.println("Value sent and returned is " +
  integerAccess1.accessInt(integerAccess1.value));
  }
}
```

Listing 2: Native code called in Listing 1

```
// Headers omitted…
JNIEXPORT jint JNICALL Java_jni_1article_1_IntegerAccess_accessInt
  (JNIEnv *, jobject, jint jInt)
{
  cout << "Value is " << jInt << endl;

  return (jInt);
}
```

Listing 3: Passing a string parameter from Java to C++

```
package jni_article_;

public class StringAccess {
  static {
    System.loadLibrary("EasyJNI");
  }
  private native String accessString(String stringValue);
  public StringAccess() {
  }

  public static void main(String[] args) {
    StringAccess stringAccess1 = new StringAccess();
    System.out.println("Value sent and returned is " +
      stringAccess1.accessString("Hello World"));
  }
}
```

Listing 4: Native code called in Listing 3

```
// Headers omitted…
JNIEXPORT jstring JNICALL
Java_jni_1article_1_StringAccess_accessString
  (JNIEnv *env, jobject jo, jstring jStringValue)
{
  jstring jStr = 0;
  const char * str =
    env->GetStringUTFChars(jStringValue, NULL); // (1)
  if (str) {
    cout << "Value is " << str << endl;
    jStr = env->NewStringUTF("Goodbye World");
    env->ReleaseStringUTFChars(jStringValue, str);
  }

  return (jStr);
}
```

Listing 5: Passing a class parameter from Java to C++.

```
package jni_article_;

public class ClassAccess {
  static {
    System.loadLibrary("EasyJNI");
  }
  private native int accessClass(Device1 devName);
  private native int accessClass(Device2 devName);
  public ClassAccess() {
  }

  public static void main(String[] args) {
    ClassAccess classAccess = new ClassAccess();
    Device2 dev2 = new Device2();
    classAccess.accessClass(dev2);
  }
}


// In device1.java
package jni_article_;

public class Device1 {
  public Device1 (int value) {
    intValue = value;
  }
  int intValue;
}

// In device2.java
package jni_article_;

public class Device2 {
  public Device2 () {
    intValue = 1;
    longValue = 11;
    stringValue = new String("Hello World");
    dev1Value = new Device1(5);
  }
  private int intValue;
  private long longValue;


  String stringValue;
  Device1 dev1Value;
}
```

Listing 6: Native code called in Listing 5

```
// Headers omitted…
// ClassAssign.AccessClass(Device2)
JNIEXPORT jint JNICALL
Java_jni_1article_1_ClassAccess_accessClass__Ljni_1article_1_Device
2_2
  (JNIEnv *env, jobject jo, jobject jOutput)
{
 // get the class of the object parameter
 jclass cls0 = env->GetObjectClass(jOutput); // (1)
 if (cls0) {
  // get the field ID of the "intValue" field of this class
  jfieldID fidInt = env->GetFieldID(cls0, "intValue", "I");
// (2)
  if (fidInt) {
   // get the value of the field in this instance of the above
class
   jint valInt = env->GetIntField(jOutput, fidInt);
   cout << "Value is " << valInt << endl;

   // get the field ID of the "longValue" field of this class
   jfieldID fidLong = env->GetFieldID(cls0, "longValue", "J");
// (3)
   if (fidLong) {
    // get the value of the field in this instance of the above
class
    jlong valLong = env->GetLongField(jOutput, fidLong);
    cout << "Value is " << (long) valLong << endl;

    // get the field ID of the String field of this class
    jfieldID fidString = env->GetFieldID(cls0, "stringValue",
"Ljava/lang/String;");                // (4)
    if (fidString) {
     // get the value of the field in this instance of the above
class

jstring valString = (jstring) env->GetObjectField(jOutput,
fidString);
     if (valString) {
         const char * str = env->GetStringUTFChars(valString,
NULL);
         if (str) {
                  cout << "Value is " << str << endl;
                  env->ReleaseStringUTFChars(valString, str);

                  // get the field ID of the Device1 object field
of this class

                  jfieldID fidObject = env->GetFieldID(cls0,
"dev1Value", "Ljni_article_/Device1;");                 // (5)

                  if (fidObject) {
                           // get a reference to this field in
this instance of the class
                           jobject valObject = env-
>GetObjectField(jOutput, fidObject);
                           if (valObject) {
                                    // get the class of the
object parameter
                                    jclass clsObject = env-
>GetObjectClass(valObject);
                                    if (clsObject) {
                                             // get the field ID
of the "intValue" field of this class
                                             jfieldID fidInt =
env->GetFieldID(clsObject, "intValue", "I");
                                             if (fidInt) {
                                                      // get the
value of the field in this instance of the class
                                                      jint
valInt = env->GetIntField(valObject, fidInt);  // (6)
                                                      cout <<
"Value is " << valInt << endl;
                                             }
                                    }
                           }
                  }
         }
      }
     }
    }
  }
 }

 return (0);
}
```

Borland Software Corp.

www.borland.com/new/jb7/5068.html

Listing 7: Functions to handle fetching string and object members of a Java

```
of a Java
// class
void AccessString(JNIEnv *env, jclass clsContaining, jobject
jContaining, LPCTSTR lpszFieldName, string & strOut)
{
 jfieldID fidString = env->GetFieldID(clsContaining, lpszFieldName,
"Ljava/lang/String;");      // get the field ID of the field of
this class
 if (fidString) {
  jstring valString = (jstring) env->GetObjectField(jContaining,
fidString);
 // get the value of the field in this instance of the above class
  if (valString) {
   const char * str = env->GetStringUTFChars(valString, NULL);
   if (str) {
    strOut = str;
    env->ReleaseStringUTFChars(valString, str);
   }
  }
 }
}

void AccessObject(JNIEnv *env, jclass clsContaining, jobject
jContaining, LPCTSTR lpszFieldName, LPCTSTR lpszFieldSig, jclass
*clsContained, jobject *oContained)
{
 // Code defensively. Initialize with "failure" values for safety
 *clsContained = static_cast<jclass>(*oContained = 0);
 // get the field ID of the field of this class
 jfieldID fidObject = env->GetFieldID(clsContaining, lpszFieldName,
lpszFieldSig);
 if (fidObject) {
  // get the value of the object field in this instance of the
above class
  *oContained = env->GetObjectField(jContaining, fidObject);
  if (*oContained) {
   // get the class of the object parameter
   *clsContained = env->GetObjectClass(*oContained);
  }
 }

 return;
}
```

Listing 8: Listing 5 modified to use the access...

```
// functions
JNIEXPORT jint JNICALL
Java_jni_1article_1_ClassAccess_accessClass__Ljni_1article_
  1_Device2_2
  (JNIEnv *env, jobject jo, jobject jOutput)
{
  // get the class of the object parameter
  jclass cls0 = env->GetObjectClass(jOutput);
  if (cls0) {
    // int and long code here...(omitted for clarity)
    string sOut;
    AccessString(env, cls0, jOutput, "stringValue", sOut);
    cout << "Value is " << sOut.c_str() << endl;

    jclass clsContained;
    jobject oContained;
    AccessObject(env, cls0, jOutput, "dev1Value",
      "Ljni_article_/Device1;", &clsContained,
      &oContained);
    if (clsContained && oContained) {
      // get the field ID of the intValue field of this
      // class
      jfieldID fidInt = env->GetFieldID(clsContained,
                                 "intValue", "I");
      if (fidInt) {
        // get the value of the int field in this instance
        // of the above class
 jint valInt = env->GetIntField(oContained, fidInt);
 cout << "Value is " << valInt << endl;
      }
    }
  }

  return (0);
}
```

Listing 9: Assigning C structures to Java classes

```
package jni_article_;

public class ClassAssign {
  static {
    System.loadLibrary("EasyJNI");
  }
  private native int assignClass(Device3 devName);
  public ClassAssign() {
  }

  public static void main(String[] args) {
    ClassAssign classAssign = new ClassAssign();
    Device3 dev3 = new Device3();
    classAssign.assignClass(dev3);
    System.out.println("Integer value is "+ dev3.intValue);
    System.out.println("Long value is " + dev3.longValue);
    System.out.println("String value is " +
      dev3.stringValue);
    System.out.println("Integer value in Device 1 is " +
      dev3.dev1Value.intValue);
  }
}
```

Listing 10: Functions to handle assignment of C++ data types to Java fields

```
Java fields
// Get an object reference to a Java object type contained within
a containing
// Java object.
jobject GetObjectReference(JNIEnv *env, jclass clsStruct, jobject
joStruct, LPCSTR lpszJavaFieldName, LPCSTR lpszSignature)
{
 // Get the field ID of the field in the Java object
 jfieldID fidField = env->GetFieldID(clsStruct, lpszJavaFieldName,
lpszSignature);
 if (fidField) {
  // get a reference to the SymAPIDirector object
  jobject joField = env->GetObjectField(joStruct, fidField);
  return (joField);
 }
 return (0);
}

/*
Assign a C int value to a Java int field.
The Java class is cls0. The Java object is oOutput. The C int
value is uiMember.
The Java class member name is lpszJavaFieldName.
Usage:
Return value:
 1 => success;
 0 => failure;
*/
int ParseIntField(JNIEnv *env, jclass cls0, jobject oOutput,
unsigned uiMember, LPCSTR lpszJavaFieldName)
{
 int iRc = 0; // failure

 // Get the field ID of the field in the Java object.
 // Note the "I" JNI signature for an int.
 jfieldID fid0 = env->GetFieldID(cls0, lpszJavaFieldName, "I");
 if (fid0) {
  // assign the (C++) int field to the Java object
  env->SetIntField(oOutput, fid0, uiMember);
  iRc = 1; // success
 }

 return (iRc);
}

/*
Assign a C long value to a Java long field.
The Java class is cls0. The Java object is oOutput. The C long
value is lCMember.
The Java class member name is lpszJavaFieldName.
Return value:
 1 => success;
 0 => failure;
*/
int ParseLongField(JNIEnv *env, jclass cls0, jobject oOutput, long
lCMember, LPCSTR lpszJavaFieldName)
{
 int iRc = 0; // failure

 // get the field ID of the field in the Java object
 jfieldID fid0 = env->GetFieldID(cls0, lpszJavaFieldName, "J");
 if (fid0) {
  // assign the (C++) long field to the Java object
  env->SetLongField(oOutput, fid0, lCMember);
  iRc = 1; // success
 }

 return (iRc);
}

/*
Assign a C single byte string value to a Java object field.
The Java class is cls0. The Java object is oOutput. The C string
is lpszCMember. The Java class member name is lpszJavaFieldName
and it's signature is lpszSignature.
Return value:
 1 => success;
 0 => failure;
*/
int ParseStringField(JNIEnv *env, jclass cls0, jobject oOutput,
LPCSTR lpszCMember, LPCSTR lpszJavaFieldName)
{
 int iRc = 0; // failure
 // convert C-string member to a java String object
 jstring str0 = env->NewStringUTF(lpszCMember);
 if (str0) {
  // get the field ID of the String field in the Java object
  jfieldID fid0 = env->GetFieldID(cls0, lpszJavaFieldName,
"Ljava/lang/String;");
  if (fid0) {
   // assign the (C++) string field to the Java object
   env->SetObjectField(oOutput, fid0, str0);
   iRc = 1; // success
  }
 }

 return (iRc);
```

Download the Code!
www.JavaDevelopersJournal.com

InetSoft
Technology Corp.

www.inetsoft.com/jdj

Jason R. Briggs J2ME Editor

The Longer-Life Pen

I've been thinking a PhD student should consider doing a thesis on the life expectancy of a pen after it's purchased. I've come up with an approximate calculation for mine: LE (Life Expectancy) =DWU (Date of Wanting to Use)-1; in other words, a pen will go missing the day before you really need to use it. There's definitely a paper there somewhere waiting to be written.

This might (or might not) lead you to wonder why someone involved in Java for mobile devices is relying on primitive, tree-killing technology like pens and paper; the short answer to this is that I am still embarrassingly PDA-less. There are, of course, a number of reasons (excuses):

1. I am still waiting for a Sharp Zaurus to review (and play with). (As you're probably aware, the Zaurus is Sharp's Linux/PersonalJava PDA.) However, it won't be a long-term solution to the problem as, short of changing my name and moving to Antarctica, I'll have to give the unit back. Perhaps the fact that I look shifty has contributed to Sharp's unwillingness to deliver…?
2. I suffer from a common IT affliction known as "What's-just-around-the-corner-itis." The main symptom of this probably genetic disorder is that whenever I find a device that might "fit the bill," so to speak, I'm immediately hit with the thought, "But wait, what about device *X*, rumored to be released in a few months? It's slightly faster and can connect directly to my cortex…"
3. In true Scrooge McDuck fashion, I don't want to dish out the cash.

On top of those three points, I've also suddenly changed my tune on the idea of convergence. Before, I looked at the idea of all-in-one devices as nice but mostly flawed in execution; now I'm starting to see movement in the right direction. Nokia's new 7210 has a color display, FM stereo radio, MMS (Multimedia Messaging) – and Java, of course. Now it's not quite a PDA, but coupled with the right Java applications, it could be close.

Nokia has another phone that also doubles as an MP3 player (the model number temporarily escapes me); if they could somehow merge that phone with the 7210 and add a pop-open cover that hides a touchscreen so you can enter data like a Palm, then I'm sure I'd purchase one on the spot.

As you can see, it's a bad case of What's-just-around-the-corner-itis.

• • •

In the "editor-eats-his-hat" department, after mercilessly "sticking the boot" into Ericsson a few months back for their lack of Java support in the T68 mobile phone (and various other models as far as I can tell), it seems the forthcoming P800/802 will include Java support. It should be a damned nice phone with an integrated digital camera, color screen, MMS, and Bluetooth support, among other features. (*Note:* It seems the new T62u also supports Java.)

• • •

On a more sober note, those of you waiting for the next installment in Bill Swaney's "Jini Surrogate As a Platform for J2ME Games" series will have to wait a bit longer than expected. Bill was recently involved in a horrific car accident, so it will probably be at least a few months before he will be able to complete the next article. Our best wishes go out to Bill for a speedy recovery.

In this month's issue, you'll find a beginner J2ME article from Fred Daoud on OO methodology for adding commands to displayable components within MIDP. Sami Lababidi from Macrospace talks about programming J2ME games. And in the tradition of do-it-yourself-electronics-in-the-garage, Bill Ray discusses controlling MP3 playback through wireless technologies – just the thing for the couch potato who can't be bothered to reach farther than the PDA on the coffee table.

jasonbriggs@sys-con.com

Author Bio

Jason R. Briggs is a Java analyst programmer and – sometimes – architect. He's been officially developing in Java for almost four years, "unofficially for five."

J2ME INDEX

QUALCOMM Incorporated

http://brew.qualcomm.com/ZJD5

Whole House Audio from the Palm of Your Hand

It's only the beginning

Part 1 of 3

WRITTEN BY
BILL RAY

In this business we often talk about how easy it is to get computers to talk to each other; computers without networks are almost inconceivable. Despite being standardized as little as five years ago, we now expect them all to play nicely together.

Even in the home, a CAT-5 connection isn't too remarkable, but mobile devices still spend most of their time in lonely isolation.

Of course, many technologies exist for connecting handheld devices to networks and other devices, but costs and limitations often cause people to question if it's really necessary: Why should a handheld be talking to the world? Well, I came up with one idea, and this article shows how I implemented it.

It starts with the conversion of my CD collection to MP3 to make it easier to find tracks and automate playback.

Once your collection is on your computer, you need software to organize it and play it back; many freely available applications will do that with varying amounts of control.

Random playback is all very well, but it can lead to some very strange combinations if your collection is broad (jumping from Beethoven to Aerosmith can be something of a shock), and it occurred to me that it should be possible to come up with something better. As the owner of a PDA (actually, several, but for the moment it's a Compaq iPaq) with a Wi-Fi card, I thought it would be nice to control my playback wirelessly.

My requirements were pretty basic; random play does work for me most of the time, but I would like to be able to pause playback and adjust the volume. There are also occasions when I'd like to select the next track to be played – when the mood takes me or when talking to guests about music – so some facility for browsing through the available tracks to select one would be good.

I should also point out that I'm very lazy, and the idea of entering the details of each recorded track doesn't appeal to me; the application must be able to cope with additional tracks being added automatically and notice if any have been removed.

Given this set of requirements, it's obviously important that all my music is stored in a standard format, something that most of the MP3 ripping software is ideally suited to. I had used an application called Audio Catalyst, but this currently has some problems with Windows 2000, so now I use AudioGrabber, which works well for my needs. This application grabs tracks from a CD, converts (rips) them to MP3 format, picks up the track listing information over the Internet, and is able to provide the MP3 file names that correspond to the track names. Because of the way I intend to use the tracks, I decided to organize them into directories by bands, then into subdirectories by album, and finally form the track name from the name of the band and the name of the track separated with a hyphen ("-"). Thus a sample listing looks something like this:

```
Blur\Park Life\Blur - Badhead.mp3
Blur\Park Life\Blur - Bank
 Holiday.mp3
Blur\Park Life\Blur - Clover Over
 Dover.mp3
… etc. …
```

Including the name of the band in the title of the file was something I was already doing so I could see who produced a song when using WinAmp or similar applications. I know there are various standards for embedding information about an MP3 file within the file itself, but I decided it would be quicker to simply use the file name, especially as this was all the information I required.

Once I decided on how the music was to be stored and what my requirements were for playing it back, I started working out what my application would look like and how it would work. A formal design process probably would have helped at this point, but since the application was only intended for my use (something that will be obvious if you care to examine the source code, which can be downloaded from the *JDJ* Web site, www.sys-con.com/java/sourcec.cfm), I felt that as long as I had the overall structure defined, I could get on with coding, something I came to regret later, as you will see.

Obviously, the application would have two major components: a server and a client. The server would have a minimum GUI, just enough to ensure it was working properly, while the client would have controls for changing the volume, pausing playback, and selecting the next track (see Figure 1).

The inclusion of a GUI on the server side was, strictly speaking, unnecessary, as it should be easy to run up a second client on the server if local control is needed. I also decided that it was important for the server and the client not to be in constant contact for two reasons: first, the battery life on an iPaq using a

Sprint PCS

http://developer.sprintpcs.com

Wi-Fi link is laughable, measured in minutes not hours (in Part 2 or 3 of this article I'll discuss moving the application over to Bluetooth with its much improved power consumption), so it was important to be able to turn the client off without affecting playback. Second, I wanted to be able to run several clients with the same server.

The motivation for this second requirement is less clear. I have several computers and a wife with her own PDA, so while the thought of being in sole control of our listening choices appealed to me, it would be less popular with others in the house.

This decision has several implications: primarily, it was not possible for the client to constantly display the track playing currently. I decided the easiest solution was to add a button to update the display on request (see Figure 2). It was possible to set up some sort of registration system for clients and establish an outgoing connection from the server to all interested clients whenever a new track started, but I decided this was too complex at the moment; it could always be added later if it seemed important.

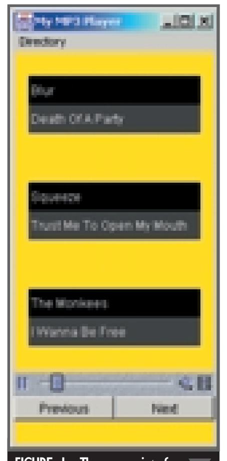


FIGURE 1 The server interface

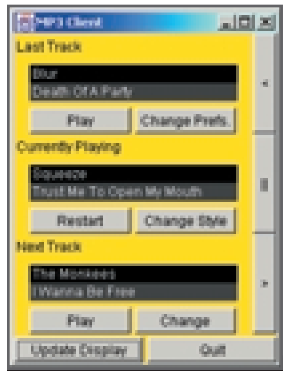
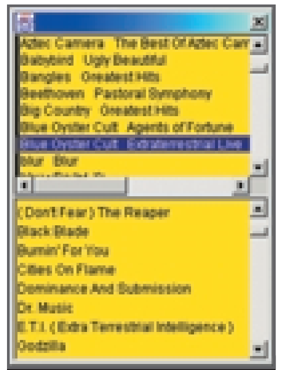


FIGURE 2 Update display on request



Aztec Camera The Best Of Aztec Cam
Babybird Ugly Beautiful
Bangles Greatest Hits
Beethoven Pastoral Symphony
Big Country Greatest Hits
Blue Oyster Cult Agents of Fortune
Blue Oyster Cult Extraterrestrial Live
Blur Blur

(Don't Fear) The Reaper
Black Blade
Burnin' For You
Cities On Flame
Dominance And Submission
Dr. Music
E.T.I. (Extra Terrestrial Intelligence)
Godzilla

FIGURE 3 Track and album list

It was clear that objects would represent the tracks, and on the server side these objects would exist in some sort of ArrayList (I like ArrayLists).

The intention was that on launch, once fed with a directory to start from, the server would scan this directory and any subdirectories looking for MP3 files that it could use to build the ArrayList. Requests to play a specific track were then indexed by its position in the ArrayList. Clearly, this meant that tracks could not be added without stopping and relaunching the server, but this was a minor point and not considered important.

The first stage was to consider the networking model: What kind of communication would take place and what kind of responses would be expected? As all communication was client-initiated, it was obvious that the server would be using a ServerSocket listening on a specific port (1710, for no other reason than it happens to be my birthday). The client would connect to that port and send a request, the server would respond, and the client would break the connection. While it's possible to send several requests and receive several responses in one go, it was not deemed necessary (this is not a very time-critical application).

On connection the server should respond with a message to confirm that it exists ("+OK"), then wait for the client to send a request for it to respond to. I adopted the Internet convention of starting all successful responding messages with a "+", while those that gave an error of any type start with a "-"; this allows the client to simply check the first character of a response to see if the request has been honored. I also decided that all network communications would be case-insensitive. While this can slow things down slightly, it makes testing easier and avoids irritating problems with the use of the wrong case. I set about mapping out all the network communication I'd be using, starting with the ability to display which tracks were being played on the client:

```
[server] +OK
[client] status
[server] +Last Playing 292#Clannad -
  Ta Me Mo Shui
[server] +Playing 866#Men Without
  Hats - 01 - In the 21st Century
[server] +Next 1057#Steeleye Span -
  The Victory
```

Note that the response not only contains the track playing currently, but also the last played and the intended next track. It also occurred to me that while Java Media Framework (JMF, the API from Sun for audio playback) does a good job of playing back MP3 files, there's a distinct lag while the track is loading and preparing to play, leading to unwanted gaps between tracks. I therefore resolved to load each track before it was needed; so while the current track is playing, the server is loading the next one in preparation. Obviously, I'll have to write the server in such a way that a track that's in the process of loading can be played, even if it means a short delay, in case the user swiftly chooses and starts a track.

The response is split with the "#" symbol, the number before it indicates the position of the track in the ArrayList held on the server. This is used to identify the track when requests are sent back to the server from the client. It was possible to use the track name, and then search the ArrayList for a matching track (and, indeed, this proved necessary in another context), but it seemed easier at the time to use numbers. There was no attempt to optimize the protocol to reduce network traffic. Given the short requests and responses this seemed unnecessary, and a human-readable protocol makes for easier testing.

The other commands are displayed in much the same way, allowing the user to pause, resume, adjust the volume, and play the next and last tracks. I did decide that the response to the status request would be useful in many other contexts, so several of the commands respond with the same list.

A request to play the previous track would lead to a lag as this track would not be prepared to play unless it maintained its prepared state, something I'd have to decide later on once I knew how long the lag was and if it was tolerable (as it turned out it's only a couple of seconds, which I can tolerate).

This covered the basic commands, but I also wanted to be able to browse the available tracks and pick one to be played, so I had to add commands to retrieve a track listing and to play a specific track indexed by number.

The track listing returns the file names of the tracks held on the server in the order in which they occur in the ArrayList (so no index number is needed). It also returns the full filename of the track. The client, on receipt of this data, can copy it into a local vector (no ArrayList in PersonalJava) with the knowledge that the index numbers will match those used on the server. The data is terminated with a full-stop (".") alone on a line, the same as the SMTP protocol.

The play request is numerically indexed; the word "play" is followed by a space and the number of the track to be played. This track then replaces the previously considered next track, and the server responds with a status response listing the last, current, and next tracks. If the intention is to start the chosen track immediately, a "next" message will need to be sent.

At this stage it's important to document your network protocol in as much detail as possible. I went to the extent of printing up examples and pinning them to the wall above my desk. Any changes to the protocol should be recorded immediately and extensions added to the written record.

Fiorano Software

www.fiorano.com/tifosi/freedownload.htm

“I can now listen to music and control what comes next, but that is only the beginning…”

When you're working with a team of programmers this is common sense, but when you're working alone it's easy to think you'll remember all the details, and then promptly come unstuck later when you discover you can't.

Having worked out what the client and the server are going to do, it was time to start writing them. Getting the JMF installed and working was very smooth, and while the API can appear confusing, it's actually easy to get the hang of it. By making the tracks responsible for their own preparation and playback, I was able to encapsulate all the JMF functions into a single class. Probably the most complex part is preparing the track for playing, which is accomplished with the following code:

```
details = new File(fname);
URL u = new URL("file:///" + fname);
Manager.setHint(Manager.PLUGIN_PLAYER,
  new Boolean(true));
p = Manager.createPlayer(u);
p.realize();
p.prefetch();
p.addControllerListener(this);
```

The String variable fname is simply the full path and filename of the MP3 file. The final line indicates that the Track class itself is the one to be informed of playback events (such as when the track finishes or has completed its preparation). Once the track is prepared, the following methods allow control from the main application:

```
  public void start() {
    p.start();
  }
  public void pause() {
    p.stop();
  }
  public void stop() {
    p.stop();
    p.setMediaTime(new Time(0));
  }
```

It's also important to be able to release the resources being used by a track for playback, which was done with the following code:

```
  public void release() {
    loaded = false;
    p.stop();
    p.deallocate();
    p.close();
  }
```

This way a track could exist in the ArrayList using few resources, then when it was known that it would be played soon, it would be prepared to start when the previous track had completed. The Boolean loaded is used to check if a track has been loaded. This is set to true on receipt of a PrefetchCompleteEvent that's passed to the ControllerListener (as specified during track preparation).

I also decided that Track objects should be responsible for deciding if they would play, so I added another method called willYouPlay(), which returns the Boolean “true” every time. This was intended to provide a more complex choice mechanism that I added later and will discuss in Part 2 of this article.

Once I got the server working, I tested it using Hyperterminal (connecting to 127.0.0.1, the TCP/IP loopback address), then set about creating a client. When developing applications such as this it's always best to start with the server, so it can be tested using a terminal application.

On the client side I developed the application using normal Java, being careful to use only constructs that are available in PersonalJava (so no beloved ArrayList).

While PersonalJava is a cutdown version of Java, as long as you're not using Swing and check before using any of the more recent APIs, you can develop in Java and then test in PersonalJava without too many problems. In this respect I did pretty well, falling down only on remembering to use elementAt when accessing a vector. The basic controls were easy enough with a simple GUI and buttons; the only issue was with picking up button click events using PersonalJava on the iPaq (the Sun implementation, though the problem also manifested itself using JEODE). Clicks don't always make themselves heard, and I was eventually forced to switch to MouseDown events to ensure usable interaction.

This had the side effect of making the buttons less responsive (they don't depress when clicked), but functionality won out over appearance.

Getting the track browsing and selection to work was considerably more effort, just working out what the interface should look like was a task in itself. I started thinking that a treeview would work well, but the thought of having to write one held me back, and I wasn't convinced that it would scale well (by this time I was working with several hundred CDs comprising over 1,600 tracks). I eventually went for two lists, both held in a single dialog frame (remember that in PersonalJava you can have only one dialog and one frame visible at a time, so the main application is in a frame and the PickTracks object is a dialog). Album names are held in the top list, while track names are displayed at the bottom (updated when a different album is selected) (see Figure 3). I decided against having any buttons in this part of the GUI as simply clicking (or tapping, as we will be using a pen) will be sufficient to select the next track to be played.

This gave me a working application. I used the PersonalJava emulation environment to make sure it was all compatible (and thus discovered the vector problem, which was easily fixed) and transferred the application to my iPaq. It works; I can now listen to music and control what comes next, but that is only the beginning…

In Part 2 of this article I'll show how I added a profile to each track, enabling me to set my mood and have suitable tracks chosen by preference. I'll also show how Bluetooth is a better network solution for this kind of application, and how it can be made to work with Java. In Part 3 I'll discuss how I added the next stage of control and enabled my mobile device to switch speakers on and off around my house.

Useful Links

- *Java Media Player:* http://java.sun.com/products/javamedia/jmf/index.html
- *Personal Java:* http://java.sun.com/products/personaljava/

bill@network23.co.uk

Author Bio
Bill Ray has worked for several telecommunications companies around Europe, including Swisscom where he was responsible for the development of their Java-compatible DTV platform. He is security editor for Wireless Business & Technology and coauthor of Professional Mobile Java Development, published by Wrox Press.

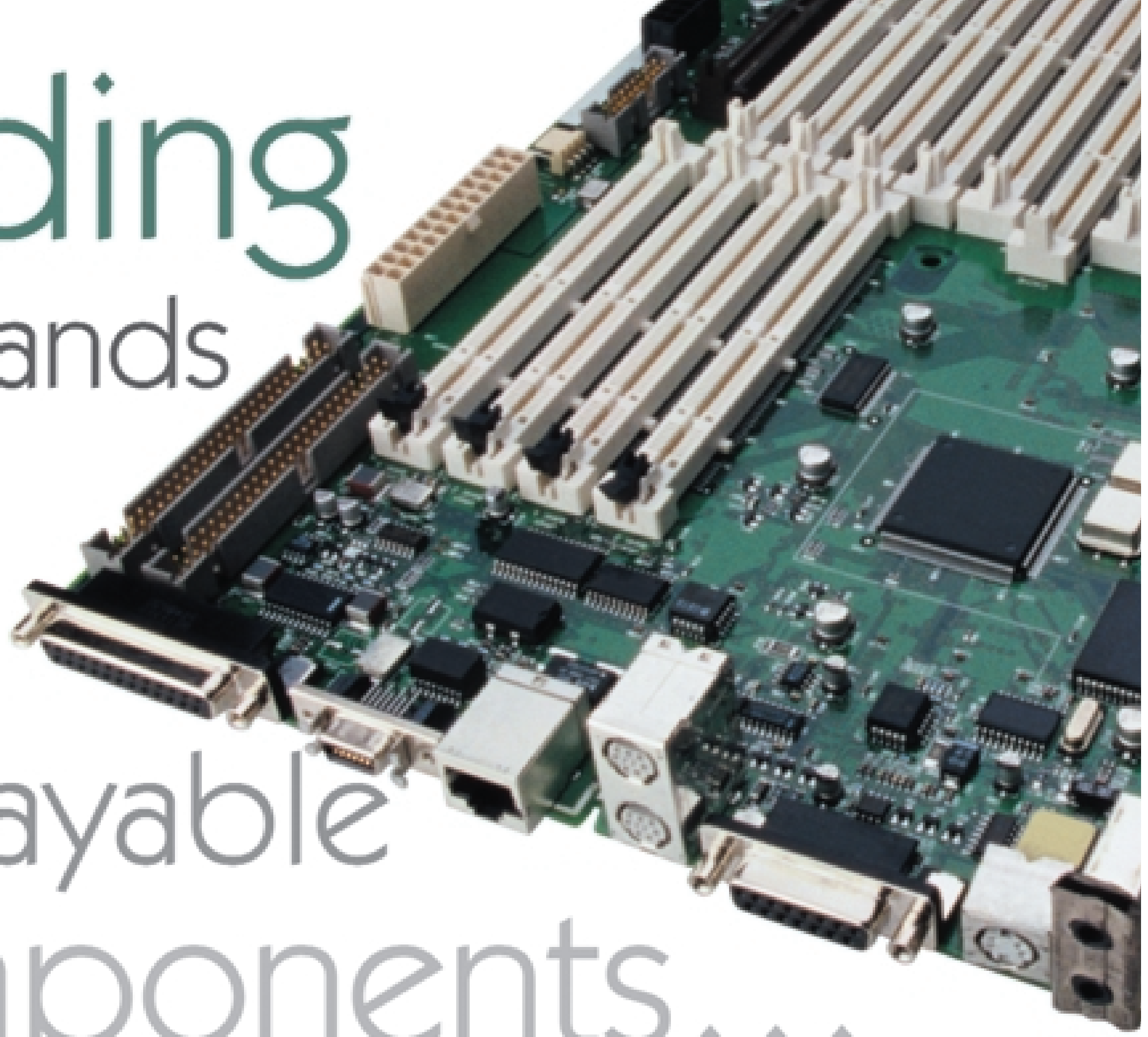
Altaworks

www.altaworks.com

Adding Commands to Displayable Components...

by Fred Daoud

...the object-oriented way



In this article I discuss which displayable components are available in J2ME and explain how commands are associated with them. In particular, I examine how to create commands, add them to displayables, and define command behavior in CommandListeners. I then demonstrate how the resulting code is typically procedural and quickly becomes cluttered when too many commands are created.

Using object-oriented principles and the Command pattern, I present two new classes that provide a simple solution to this problem. These classes provide structure and reusability to the process of creating commands for displayables, resulting in code that's cleaner and easier to maintain.

Overview of Displayable Components

In the Mobile Information Device Profile (MIDP) there's no AWT (and certainly no Swing) for creating graphical user interfaces. Instead, GUIs are created using classes from the javax.microedition.lcdui package. As shown in Figure 1, the abstract Displayable class is the superclass from which you create objects that can be displayed on the screen of a MIDP device.

The two immediate subclasses of Displayable are Canvas and Screen. The Canvas class creates GUI components that use low-level graphical primitives to draw themselves. To use the Canvas class you must subclass it and implement its paint() method, which receives a Graphics parameter. The Graphics class, similar to its J2SE counterpart, provides methods to draw lines, rectangles, arcs, strings, and images.

The Screen class, on the other hand, creates high-level GUI components. Like the Canvas class it's abstract, but it has four direct subclasses that are concrete and ready to use: Alert, List, TextBox, and Form. All but the Form, which is quite blank on its own, are shown in Figure 2. The Alert class displays a message similar to a dialog box. The List presents choices using one of the following schemes: EXCLUSIVE, MULTIPLE, or IMPLICIT. The TextBox allows the user to enter text and can optionally restrict the input format; for example, permit numbers only.

The Form class, unlike its three siblings, is not a specific GUI component; rather it serves as a container for the other components, making it possible to build more complex user interfaces. The components that the Form contains are subclasses of the Item class. These include the ChoiceGroup (equivalent to the List), DateField, Gauge, ImageItem, StringItem, and TextField. There are no layout managers in MIDP; instead, the device automatically handles the layout, traversal, and scrolling of the items that are added to a Form. In most cases, the components are laid out vertically.

Besides having graphical representation, displayable components can also have

Pramati Technologies

www.pramati.com

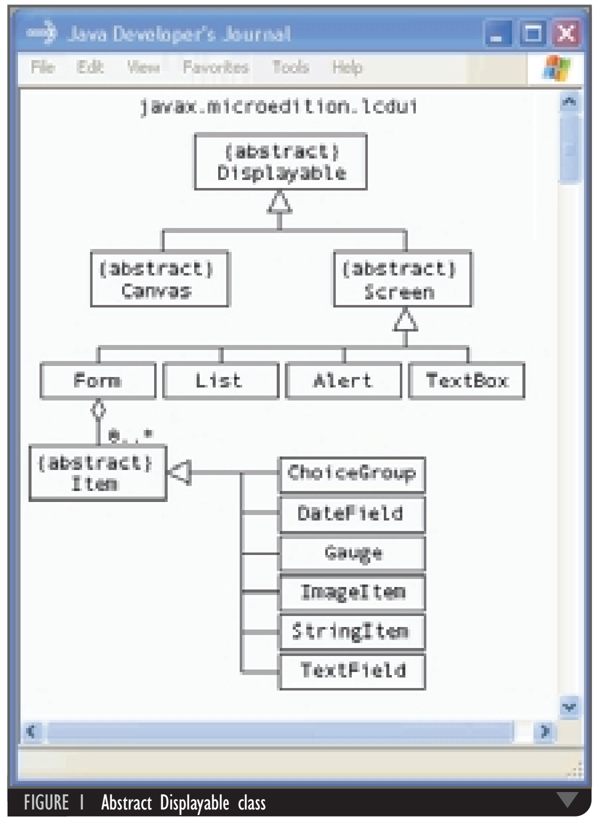


FIGURE 1 Abstract Displayable class

additional behavior associated with them. To do this, add commands and set a CommandListener that will be notified when the commands are invoked.

> "Besides having graphical representation, displayable components can also have additional behavior associated with them"

Commands and CommandListeners

Figure 3 shows the relationship between the Displayable class, commands, and CommandListeners. Notice that multiple commands may be added to a displayable, but only one CommandListener may be set. I'll examine some of the consequences of this later. For now, let's look at how to create commands and CommandListeners.

To create a command, specify the label string, type, and priority of the command. The label string represents the command in the user interface. The command type gives a hint to the device on the semantics of the command. It's up to the device implementation to decide which scheme to use for commands of a certain type. The defined command types are BACK, CANCEL, EXIT, HELP, ITEM, OK, SCREEN, and STOP. And the priority is an integer that indicates the importance of the command with respect to other commands. Commands with a smaller number are given a higher priority. When adding more commands than the maximum that the device can display at one time, commands with a higher priority will be shown and those of a lower priority will be available only by other means (a menu, for example).

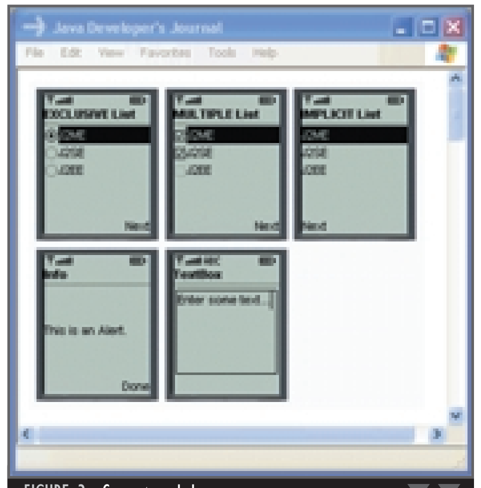


FIGURE 2 Concrete subclasses

After creating a command, add it to the displayable by using its addCommand() method. For example:

```
Command command = new Command("Go",
  Command.SCREEN, 0);
displayable.addCommand(command);
```

The Command class is not where you define what happens when the command is invoked. Instead, you must create a class that implements the CommandListener interface, which contains one method:

```
public void commandAction(Command c,
  Displayable d);
```

This method is called when a command that was added to a displayable is invoked, so this is where you place the behavior for your commands. After creating your CommandListener, associate it with the displayable by using the setCommandListener() method. Notice that the method is not addCommandListener() as would have been the norm in J2SE. As stated earlier, in J2ME there can be only one CommandListener per displayable. This means that the behavior for every command of a displayable must be defined in one implementation of CommandListener.

Procedural Example

I'll use an example to illustrate the use and consequences of the current Command–>CommandListener process. Consider a simple MIDlet that displays the three different types of Lists: EXCLUSIVE, MULTIPLE, and IMPLICIT. The MIDlet displays an empty Form with four commands: one for each type of List, and an Exit command.

When the user selects a type, a List of that type with arbitrary choices is displayed. The user interacts with the List and can select OK or Cancel. OK displays a confirmation of the selected List item(s), while Cancel goes back to the Form that offers the List type choices.

Listing 1 shows the code for this example. In line 6, I create the initial Form. Lines 8–11 define the items that will be displayed in each List and a Boolean array that will contain flags indicating the selected items. Lines 13–20 create the four commands for the Form.

Lines 22–25 define the OK and the Cancel commands for the Lists.

Rational User Conference 2002

www.rational.com/ruc

In the startApp() method, I add the four commands to the Form and set its CommandListener, defined on lines 44–69. In this code, note that I determine the List type according to the invoked command. I then create a List of that type and add the OK and Cancel commands. Finally, I set the CommandListener, defined in lines 71–100, and display the List.

Note that line 79 tests for the OK command as well as the special List.SELECT_COMMAND; this command is invoked when the user makes a selection on a List of type IMPLICIT, so we must test for it.

Listing 1 demonstrates the process of adding commands to displayables and associated CommandListeners to define the behavior for the commands. However, defining the behavior separate from the command, combined with the restriction of having only one listener per displayable, yields procedural code that prevents us from leveraging OO principles. Resulting problems include:

- The CommandListener's commandAction() method quickly becomes monolithic and littered with if/else statements.
- If you dynamically remove a command from a displayable, the corresponding conditional statement in the commandAction() method remains and will needlessly be tested when other commands are invoked.
- The CommandListener is tightly coupled with the commands that it services. To add another command to a displayable, you must not only add the code for the command, but also modify the CommandListener.
- The command and its behavior are defined in two separate places in the code, hindering readability.
- You can't create objects that define the behavior for your commands. So you can't take advantage of features that OO provides, like inheritance and polymorphism.

After creating a few MIDlets, I became frustrated by these problems in the command creation process. There had to be a better way.

"After creating a few MIDlets, I became frustrated by these problems in the command creation process. There had to be a better way"

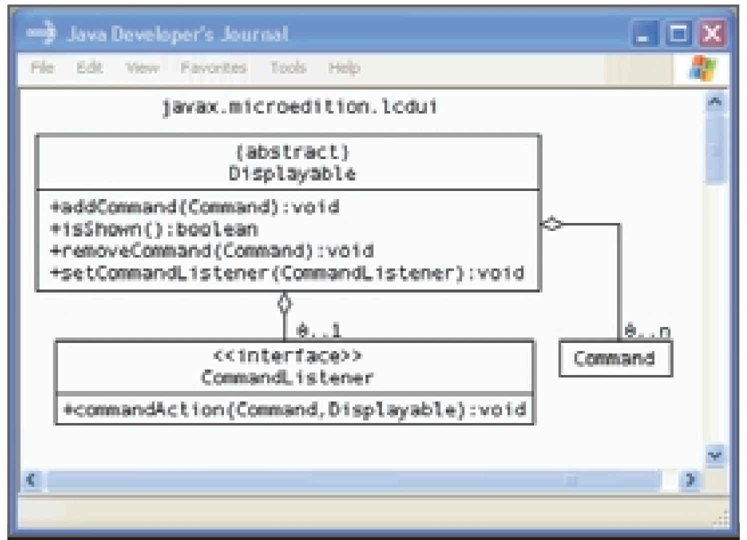

Java Developer's Journal
File Edit View Favorites Tools Help
javax.microedition.lcdui
{abstract}
Displayable
+addCommand(Command):void
+isShown():boolean
+removeCommand(Command):void
+setCommandListener(CommandListener):void
0..1
0..n
<<interface>>
CommandListener
+commandAction(Command,Displayable):void
Command

FIGURE 3 Displayable class, commands, and CommandListeners

Command Pattern and CommandAction

Creating commands and their behavior the OO way and solving the problems enumerated in the previous section meant opening my copy of *Design Patterns* (see Resources section) and reviewing the Command pattern. Its intent is to encapsulate a request as an object by providing a method that will be invoked to execute the command. *Refactoring* (see Resources) confirms this by identifying a series of if/else statements (or switch statements) as a "bad smell" in the code that should be refactored. The proposed solution is to "Replace Conditional with Polymorphism": move each leg of the conditional to an overriding method in a subclass and make the original method abstract.

To do this, I created the CommandAction class, which contains a command and provides the aforementioned method:

```
public abstract void execute(Displayable d);
```

This method makes it possible to associate the behavior of the command with the command itself. For convenience, the CommandAction class (see Listing 2) provides two constructors, one that accepts a command and one that accepts the label, type, and priority and creates a command from those parameters. Now, the command and its behavior are encapsulated in an object. For example:

```
private CommandAction goCommand
 = new CommandAction("Go", Command.SCREEN, 0)
{
  public void execute(Displayable d) {
    // behavior code for the Go command
  }
};
```

The CommandManager Class

The second class I needed was one that accepts CommandActions, adds the commands they contain to displayables, and invokes their execute() method when the command is invoked. I called this class CommandManager. It provides a method to add a CommandAction to a displayable:

```
public void add(CommandAction cmdAction,  Displayable dis
  playable);
```

In your MIDlet, create a single CommandManager that will handle all CommandActions for all displayables. A remove method is also provided:

```
public void remove(CommandAction cmdAction, Displayable
  d);
```

In the CommandManager class, I use three Hashtables, one to answer each of the following questions:

- ***Hashtable 1:*** Given a displayable, how many commands have been added to it?
- ***Hashtable 2:*** Given a command, what is its corresponding CommandAction?
- ***Hashtable 3:*** Given a CommandAction, to how many displayables has it been added?

Jinfonet

www.jinfonet.com

When a CommandAction is added to a displayable, the CommandManager adds the command contained by the CommandAction to the displayable and increments the counter for that displayable in Hashtable 1. If this counter, after being incremented, is 1, the CommandManager registers itself as the CommandListener for this displayable. If the counter is greater than 1, the CommandListener is already registered.

Similarily, every time a CommandAction is removed from a displayable, the CommandManager removes the corresponding command from the displayable and decrements the counter. If this counter goes back to 0, the CommandManager removes itself as a CommandListener for that displayable by calling setCommandListener(null). The entry is also removed from Hashtable 1.

Hashtable 2 is necessary because the commandAction() method receives a command parameter and we must determine the corresponding CommandAction in order to call its execute() method.

Hashtable 3 tells us whether upon removing a command, we can remove the Command–>CommandAction mapping from Hashtable 2. Since it's possible that the same command is used for more than one displayable, we must make sure that no displayables use the command before removing the entry from Hashtable 2.

The complete code for the CommandManager class is shown in Listing 3. *Note:* The Hashtables use 1-element int arrays for counters because they require object values; when using integer objects you have to create new integers each time, which results in a lot of garbage to be collected.

Benefits

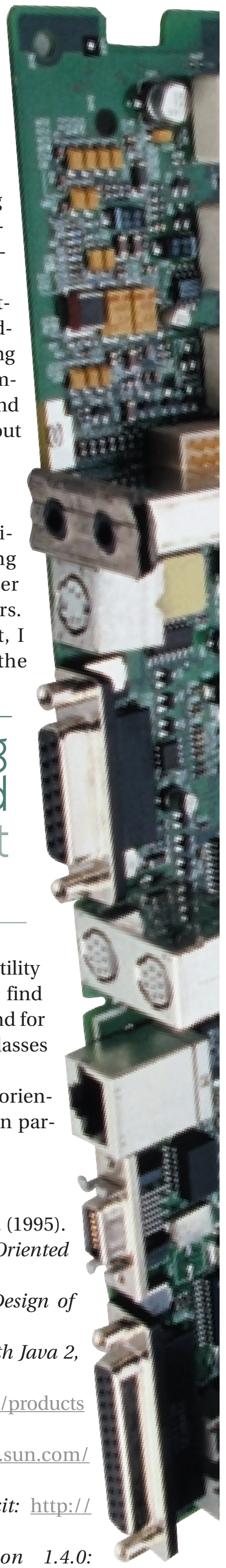
I find that using the CommandAction and CommandManager classes yield benefits at all stages of coding: creation, modification, and maintenance. These benefits are the results of making the process more object-oriented. The key is to encapsulate a command and its behavior in a single class. Having the two close together in code also improves readability.

The CommandManager is a helper class that defines once and for all the mechanism of adding and removing commands to displayables and setting the CommandListener. This reuse reduces clutter in client code.

OO principles can now be put to use when creating commands. Since commands and CommandListeners are loosely coupled, adding or removing commands does not require changes to a CommandListener. Your code becomes more modular, and you can add new commands to your system without having to modify or recompile existing classes.

Conclusion

J2ME is an exciting platform for writing applications that run on wireless devices. After writing several MIDlets, I realized that there was a better way to create commands and CommandListeners. Since these are used in just about every MIDlet, I felt it was worthwhile to carefully consider the issue.



"OO principles can now be put to use when creating commands. Your code becomes more modular, and you can add new commands to your system without having to modify or recompile existing classes"

Object-Oriented Example

Let's now implement the previous example using the CommandAction and CommandManager classes (see Listing 4).

Line 7 creates the CommandManager that will be used for all commands of all displayables.

Now, instead of having the if/else statements to determine which type of List the user chose, we can define a class that extends CommandAction and constructs a command and its behavior based on the List type (lines 14–27).

Lines 29–34 define the Exit command.

Remember that the OK command can be directly invoked by the user, but the special List.SELECT_COMMAND can also be invoked when the List type is IMPLICIT. Lines 36–60 define the OK command as a subclass of CommandAction because we have to create two types, one for the OK command and one for the List.SELECT_COMMAND. For convenience, I provided both of the CommandAction constructors.

Lines 62–67 define the Cancel command.

Lines 70–73 show how easy it is to add the List CommandActions to the Form using the CommandManager. You don't have to bother with CommandListeners; the command behavior is defined in the CommandAction. Similarily, lines 20–22 add the OK and Cancel CommandActions to the List.

When a situation like this arises, add some utility classes to help you program using the design you find most advantageous. Take the time to do this once and for all; you'll be glad you did when you're using those classes over and over again.

I hope I've encouraged you to strive for object-orientation in general and to continue exploring J2ME in particular.

Author Bio
Fred Daoud is an independent contractor and a Sun Certified Java developer. He uses J2SE, J2EE, design patterns, and OO principles to develop networked GUI applications. Fred holds a computer engineering degree from McGill University in Montreal.

Resources

- Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley.
- Fowler, M. (2000). *Refactoring: Improving the Design of Existing Code*. Addison-Wesley.
- Knudsen, J. (2001). *Wireless Java: Developing with Java 2, Micro Edition*. Apress.
- *Java 2 Platform, Micro Edition:* http://java.sun.com/products/j2me
- *Mobile Information Device Profile:* http://java.sun.com/products/midp
- *Java 2 Platform Micro Edition, Wireless Toolkit:* http://java.sun.com/products/j2mewtoolkit
- *Java 2 Platform, Standard Edition, version 1.4.0:* http://java.sun.com/j2se/1.4

fred@javelotinc.com

Quintessence Systems Limited

www.in2j.com

Listing 1

```
import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;

public class ListDemo1 extends MIDlet {
  private Display _display;
  private Form _form = new Form("Choose a type of List:");

  private static final String[] _items
    = {"J2ME", "J2SE", "J2EE"};
  private static final boolean[] _flags
    = new boolean[_items.length];

  private Command _exCmd = new Command("EXCLUSIVE",
    Command.SCREEN, 0);
  private Command _multCmd = new Command("MULTIPLE",
    Command.SCREEN, 0);
  private Command _impCmd = new Command("IMPLICIT",
    Command.SCREEN, 0);
  private Command _exitCmd = new Command("Exit",
    Command.EXIT, 0);

  private Command _okCmd = new Command("OK",
    Command.SCREEN, 0);
  private Command _cancelCmd = new Command("Cancel",
    Command.SCREEN, 0);

  public void startApp() {
    _form.addCommand(_exCmd);
    _form.addCommand(_multCmd);
    _form.addCommand(_impCmd);
    _form.addCommand(_exitCmd);
    _form.setCommandListener(_formListener);

    _display = Display.getDisplay(this);
    _display.setCurrent(_form);
  }

  public void pauseApp() { }

  public void destroyApp(boolean unconditional) {
    notifyDestroyed();
  }

  private CommandListener _formListener
  = new CommandListener() {
    public void commandAction(Command cmd,
    Displayable displayable) {
      if (cmd == _exitCmd) {
        notifyDestroyed();
        return;
      }
      int listType = -1;
      if (cmd == _exCmd) {
        listType = Choice.EXCLUSIVE;
      }
      else if (cmd == _multCmd) {
        listType = Choice.MULTIPLE;
      }
      else if (cmd == _impCmd) {
        listType = Choice.IMPLICIT;
      }
      List list = new List("Select an item:",
        listType, _items, null);
      list.addCommand(_okCmd);
      list.addCommand(_cancelCmd);
      list.setCommandListener(_listListener);
      _display.setCurrent(list);
    }
  };

  private CommandListener _listListener
  = new CommandListener() {
    public void commandAction(Command cmd,
    Displayable displayable) {
      List list = (List) displayable;
      if (cmd == _okCmd || cmd == List.SELECT_COMMAND) {
        list.getSelectedFlags(_flags);
        StringBuffer msg
          = new StringBuffer("You selected:");
        for (int i = 0; i < _flags.length; i++) {
          if (_flags[i]) {
            msg.append("\nItem ");
            msg.append(String.valueOf(i + 1));
            msg.append(" - ");
            msg.append(list.getString(i));
          }
        }
        Alert alert = new Alert("Info",
          msg.toString(), null, AlertType.INFO);
        alert.setTimeout(Alert.FOREVER);
        _display.setCurrent(alert);
      }
      else if (cmd == _cancelCmd) {
        list.removeCommand(_okCmd);
        list.removeCommand(_cancelCmd);
        list.setCommandListener(null);
        _display.setCurrent(_form);
      }
    }
  };
}
```

Listing 2

```
import javax.microedition.lcdui.*;

public abstract class CommandAction {
  private Command _command;

  public CommandAction(String label, int type,
  int priority) {
    _command = new Command(label, type, priority);
  }
  public CommandAction(Command command) {
    _command = command;
  }

  public Command getCommand() {
    return _command;
  }
  public void setCommand(Command command) {
    _command = command;
  }

  public abstract void execute(Displayable d);
}
```

Listing 3

```
import javax.microedition.lcdui.*;
import java.util.*;

public class CommandManager implements CommandListener {
  private Hashtable _dToC = new Hashtable();
  private Hashtable _cToD = new Hashtable();
  private Hashtable _commands = new Hashtable();

  public void add(CommandAction cmdAction,
  Displayable displayable) {
    int[] cmdCount = increment(_dToC, displayable);
    if (cmdCount[0] == 1) {
      displayable.setCommandListener(this);
    }
    increment(_cToD, cmdAction);

    Command command = cmdAction.getCommand();
    displayable.addCommand(command);
    _commands.put(command, cmdAction);
  }

  public void remove(CommandAction cmdAction,
  Displayable displayable) {
    Command command = cmdAction.getCommand();
    displayable.removeCommand(command);

    int[] cmdCount = (int[]) _dToC.get(displayable);
    cmdCount[0]--;
    if (cmdCount[0] == 0) {
      _dToC.remove(displayable);
      displayable.setCommandListener(null);
    }

    int[] dispCount = (int[]) _cToD.get(cmdAction);
    dispCount[0]--;
    if (dispCount[0] == 0) {
      _commands.remove(command);
    }
  }

  public void commandAction(Command command,
  Displayable displayable) {
    CommandAction cmdAction
      = (CommandAction) _commands.get(command);
    if (cmdAction != null) {
      cmdAction.execute(displayable);
    }
    else {
      System.err.println("No command action for command!");
    }
  }

  private int[] increment(Hashtable ht, Object target) {
    int[] count = (int[]) ht.get(target);
    if (count == null) {
      count = new int[1];
      ht.put(target, count);
```

CTIA WIRELESS I.T.

www.ctiashow.com

```
      }
      count[0]++;
      return count;
   }
}
```

Listing 4

```
import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;

public class ListDemo2 extends MIDlet {
   private Display _display;
   private Form _form = new Form("Choose a type of List:");
   private CommandManager _cm = new CommandManager();

   private static final String[] _items
      = {"J2ME", "J2SE", "J2EE"};
    private static final boolean[] _flags
      = new boolean[_items.length];

    private class ListCommand extends CommandAction {
      private List _list;
      public ListCommand(int type, String label) {
         super(label, Command.SCREEN, 0);
         _list = new List("Select an item:",
           type, _items, null);
         _cm.add(new OkCommand("OK", Command.SCREEN, 0), _list);
         _cm.add(new OkCommand(List.SELECT_COMMAND), _list);
         _cm.add(_cancelCmd, _list);
      }
      public void execute(Displayable displayable) {
         _display.setCurrent(_list);
      }
    }

    private CommandAction _exitCmd = new CommandAction(
    "Exit", Command.EXIT, 0) {
      public void execute(Displayable displayable) {
       notifyDestroyed();
      }
    };

    private class OkCommand extends CommandAction {
      public OkCommand(String label, int type, int priority) {
         super(label, type, priority);
      }
      public OkCommand(Command command) {
         super(command);
      }
      public void execute(Displayable displayable) {
         List list = (List) displayable;
         list.getSelectedFlags(_flags);
         StringBuffer msg = new StringBuffer("You selected:");
         for (int i = 0; i < _flags.length; i++) {
           if (_flags[i]) {
             msg.append("\nItem ");
             msg.append(String.valueOf(i + 1));
             msg.append(" - ");
             msg.append(list.getString(i));
           }
         }
         Alert alert = new Alert("Info",
           msg.toString(), null, AlertType.INFO);
         alert.setTimeout(Alert.FOREVER);
         _display.setCurrent(alert);
      }
    };

    private CommandAction _cancelCmd = new CommandAction(
    "Cancel", Command.SCREEN, 0) {
      public void execute(Displayable displayable) {
         _display.setCurrent(_form);
      }
    };

    public void startApp() {
      _cm.add(new ListCommand(Choice.EXCLUSIVE, "EXCLUSIVE"), _form);
      _cm.add(new ListCommand(Choice.MULTIPLE, "MULTIPLE"), _form);
      _cm.add(new ListCommand(Choice.IMPLICIT, "IMPLICIT"), _form);
      _cm.add(_exitCmd, _form);

      _display = Display.getDisplay(this);
      _display.setCurrent(_form);
    }

    public void pauseApp() { }

    public void destroyApp(boolean unconditional) {
      notifyDestroyed();
    }
}
```

Download the Code!
www.JavaDevelopersJournal.com

SYS-CON Media

www.sys-con.com

iSavvix

www.community.isavvix.com

Programming Games in J2ME

The battle for market share

WRITTEN BY
SAMI LABABIDI

J2ME's Mobile Information Device Profile (MIDP) promises to become one of the most widespread computing platforms over the next few years as an increasing number of mobile phones include a Java Virtual Machine.

Given the popularity of preinstalled games on mobiles, Java games are likely to captivate a large audience. The first commercial download services for MIDP applications or "MIDlets" have already been launched and the market is set to grow at an explosive rate. Nokia alone expects to sell more than 50 million of these devices by the end of 2002, so the opportunities associated with writing good games for Java-enabled devices are staggering

However, while learning the basics of MIDP is simple enough for most developers, understanding the constraints of the platform and the tricks to get around them is crucial for efficient development. Creating compelling content, especially the type that uses the mobile network, involves a lot of trial and error. This article is a collection of lessons I've learned from coding games for the first generation of J2ME devices.

The key factor to consider when designing a MIDP game is your target audience and hence the set of target devices. If your MIDlet needs to run across devices with varying performance and graphical abilities, you need to take this into account in the design of your code. This is especially important when it comes to the use of libraries proprietary to a handset manufacturer, the use of graphics, and CPU-intensive AI calculations. If the aim is to capture the mass market, the game needs to be playable on the lowest common denominator device – currently a 64x95 pixel 1-bit screen, a slow, low-end CPU, and less than 100KB of heap memory. Modifying the code to cater to high-end devices should then ideally be a matter of modifying only a clearly defined portion of the application. It pays to spend some additional time on design. For example, a game written using the full Siemens game API for the 101x80 pixel screen of the SL45i will be very difficult to port to the Nokia 6310i if portability has not been designed into the code from day one.

One of the most notable shortcomings of game development in MIDP 1.0 is its lack of advanced graphical features. The Graphics class does not provide any form of transparency, whether you're using an image mask or defining a transparent color. There's also no way to read or write to individual pixels, which prevents even basic image manipulations, such as fading an image to black.

These limitations may not seem too serious at first, as the obvious way around them would be to read in local image files directly using the "javax .microedition.io" package and write decoders of JPEG or GIF or other image file formats. You could then map out the bytes of the image manually and overcome the above limitations. While this would result in full control over the image pixels, it's prohibitively slow. MIDP does not provide a fast pixel plotting function and there's no quick way of getting the resulting image buffer painted onto the screen. In fact, the only option is to draw lines 1 pixel in length with the drawLine() function. Filling a 100x100 screen with 10,000 drawLine() calls on a mobile device is just not feasible with near future device performance!

Most device manufacturers have responded to the limitations of the native graphics in J2ME by adding additional classes in their own proprietary API. However, it's dangerous to rely on these, as it breaks the portability of the code and is likely to result in lengthy and frustrating modifications to the code for different manufacturers' handsets. Generating an abstraction layer is, of course, an option, but with a maximum JAR file size of 30KB for typical low-end handsets, this type of layer must be extremely light. MIDP NG (Next Generation), scheduled for public review in the JCP in the near future, will fix some of these problems. Similarly, the Open Mobile Architecture initiative driven by Nokia promises much, but it remains to be seen whether it will be enough to make handset manufacturers converge on APIs.

When programming fast-moving games, there are a few additional points to keep in mind. Anyone who has done game programming knows that double buffering is an important tool to achieving smooth and "tear-free" animations. The implementation of the Graphics class is double-buffered on many devices, as can be seen by a quick call to the Canvas.isDoubleBuffer() method.

This means that any graphics drawing functions are buffered off-screen before they're displayed on-screen. However, as double buffering usually refers to switching between one back-screen buffer and one front-screen buffer, not buffers for individual Graphics calls, this has been the source of some confusion. Since all the screen paintings are done within the paint (Graphics g) method, it's not hard to imagine what happens when there are many Graphics drawing calls and each drawing call is displayed onto the screen one by one.

It's not difficult to write a double-buffering system. First set up an off-screen buffer in the form of a "javax.microedition.lcdui.Image", then do every drawing operation using the Graphics obtained from this buffer instead of from the actual screen. After you're done with the drawing, emulate

SYS-CON Media

www.sys-con.com

the "flipping" of the buffers by drawing the whole offscreen buffer onto the actual screen in one go with one drawImage() call (see Listing 1).

Another issue to take into account is CPU speed and speed throttling. Since speed can vary markedly between devices, speed throttling should be applied to any games with animations. Some people calculate their own timed delays in their animation thread, while others prefer simply using the "java.util.Timer" to schedule a "java.util .TimerTask" for all the animation duties. Depending on the type of game, you may want to try scheduling the Timer to go off with a 50–100ms delay (10–20 fps).

Designing Networked J2ME Games

The prospect of providing multiplayer games for mobile phones is exciting. However, before you write your first line of code for your multiplayer first person shooter, keep in mind some important limitations. A good place to start is to look at the constraints of the network and what is possible using the J2ME communications library.

The oft-cited limitations in the bandwidth of current 2G and 2.5G networks are an important bottleneck for networked games. However, lesser-known but equally significant constraints are packet latency and variability in network performance. A limited bandwidth implies that communication over the network should be reduced to a minimum, while packet latency and variability mean that the code must run and be playable even in vastly variable network conditions. The problems are compounded by the implementation of network communication in MIDP 1.0.

All J2ME devices implement HTTP connections and some also offer a socket API. The package containing the communication classes is "javax.microedition.io", with the most important classes being Connector, Connection, and HttpConnection. The best way to communicate with a Web server is to use POST requests. Listing 2 shows an outline of an HTTP POST connection.

Alternatively, Listing 3 provides an outline of a sample socket connection.

The J2ME communication API has some notable limitations. It does not implement "java.io.Serializable" and neither does it support HTTP session functionality. If these features are required, there's no option but to reimplement them.

Figure 1 provides an overview of the elements in a networked J2ME game architecture using either GSM CSD or GPRS as a carrier. Relatively good performance can be achieved even on a GSM network, if the size of the message transmitted is kept to a minimum.

However, GPRS is the carrier of choice, as the user doesn't need to wait to connect and pays only for the data transmitted. While bandwidth is unlikely to be a problem, achieving an acceptable average latency over the network can pose a significant challenge. A round-trip time of 10 seconds over a CSD call on GSM and 4 seconds over GPRS can be considered average even with some optimizations. This limits games to the turn-based type that utilize only current network infrastructure. Real-time multiplayer games will become a reality only when 3G networks promising sub-100ms latencies are launched.

Despite the limitations, there are many things you can do with the network. Shared high scores or level downloads are a good example. Another is the possibility of playing against advanced game servers. Because of hardware limitations it's sometimes not feasible to develop a sufficiently powerful AI on mobile devices. Game servers have a vast advantage in computing power, which makes utilizing a server-side AI a compelling proposition. Take a scenario where the user plays locally and the AI uses a minimax alpha beta algorithm with a depth of two. On low-end devices this takes between 20 seconds and 2 minutes to calculate, and the game is hardly playable. Compare that to a scenario in which the user plays against a remote game server where the AI uses the same algorithm with a more powerful search depth of four. The server computes the move instantaneously, it takes 4–8 seconds to get the move from the server, and the game is much more playable.

The game server itself can be implemented as an HTTP server or a socket server. A socket server is faster than an HTTP server, as there's no HTTP overhead, but it can be used only for those devices that support sockets. In addition, using HTTP helps the long-term scalability of the service, as the number of active devices polling for moves and messages increases. As the client is written in Java, it's easy to connect to servlets and to implement the server in Java: the HTTP requests coming from J2ME devices are exactly the same as other HTTP requests. The standard Java Serialization API cannot be used in this case, but it's very easy to write your own. XML is a standard way to communicate with other types of Web servers like PHP or CGI, but it's obviously much less efficient.

Using a standard Web server like Apache/Tomcat or Jetty has the advantage of leveraging the scalability that has already been achieved for other Web/WAP applications. However, due to the nature of the wireless client and the wireless network infrastructure, some modifications need to be made in order to optimize the data traffic, as Web server providers have yet to fully adapt to the wireless market. In the short term, the use of open source products may be a good solution.

An even better solution is to use a Java application server as a back end. EJB/JMS/J2ME integration is a very active topic at the moment and this is definitely the way forward to ensure speed, availability, fault tolerance, and quality of service for all networked wireless applications, including multiplayer games. This will also require some tweaking in these early days. For example, a combination of J2ME, Jetty, and JBoss is a good open source (nearly) 100% Java solution.

These are just a few technical issues to consider in the development of MIDP games. Many times an equally challenging task is handling the creative and business side of MIDP game development. With so many game ideas already exhausted and brands becoming increasingly important in the mobile marketplace, coding a clone of an '80s hit is no longer enough to convince an operator or download portal to resell your application.

A solid technology base and a good creative team is a must for a successful company in the MIDP games arena. But as in all things, it's on the business side where the battles for market share will eventually be won and lost.

sami@macrospace.com

Author Bio
Sami Lababidi is chief technology officer at Macrospace, Ltd., a provider of J2ME solutions. Sami has focused on J2ME since its inception and has developed and deployed J2ME products and services throughout Europe.

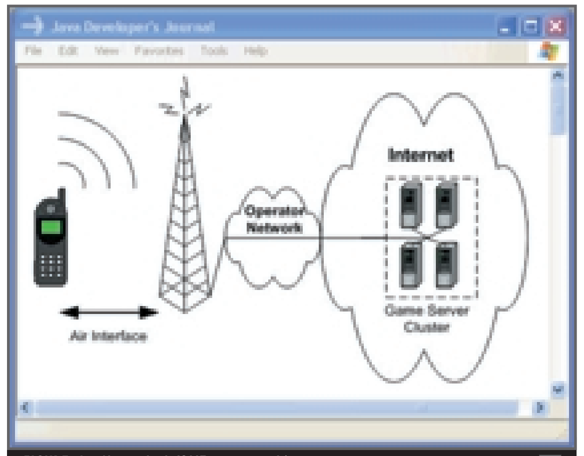


FIGURE 1 Networked J2ME game architecture

Simplex Knowledge Company

skc.com

Listing 1

```
javax.microedition.lcdui.*;

public class gameCanvas extends Canvas
{
    …
    Image    backBuffer;
    Graphics     backBufferGraphics;
    …
    public gameCanvas()
    {
        ….
        // initialize the back-buffer to be the same dimension as
         the canvas,
        // and get the Graphics of the buffer for off-screen
                  drawing
        backBuffer = Image.createImage( getWidth(), getHeight() );
        backBufferGraphics = backBuffer.getGraphics();
        ….
    }
    ….
    public void paint(Graphics g)
    {
        …
        // do all the Graphics drawing onto the back-buffer
        backBufferGraphics.draw…
        …
        // now "flip" the back-buffer to the front
        // by drawing the whole back-buffer image onto the
                  screen.
        // since individual Graphics call is "double-buffered",
        // this won't create any "tearing" on the screen.
        g.drawImage( backBuffer, 0, 0 Graphics.TOP|Graphics.LEFT );
    }
    ….
}
```

Listing 2

```
HttpConnection c = null;
InputStream is = null;
OutputStream os = null;
try
{
   // open http connection
c = (HttpConnection)Connector.open( url, Connector.READ_WRITE );

 // Set the request method and headers
 c.setRequestMethod( HttpConnection.POST );
 c.setRequestProperty( "User-Agent" , "Profile/MIDP-1.0
 Configuration/CLDC-1.0" );
 c.setRequestProperty( "Content-Language" , "en-US" );
 c.setRequestProperty( "Connection" , "close" );

 // write request
 os = c.openOutputStream();
 ....

 // get response
 is = c.openInputStream();
 ...

 ...
}
finally
{
  try{ os.close(); }catch( Exception e ){}
  try{ is.close(); }catch( Exception e ){}
  try{ c.close(); }catch( Exception e ){}
}
```

Listing 3

```
StreamConnection connection = null;
InputStream is = null;
OutputStream os = null;
try
{
  Connection aConnection = Connector.open( "socket://" + host +
  ":" + port);
  connection = (StreamConnection) aConnection;
  os = connection.openOutputStream();
  ..
  is = connection.openInputStream();
  ...
}
finally
{
  try{ os.close(); }catch( Exception e ){}
  try{ is.close(); }catch( Exception e ){}
  try{ c.close(); }catch( Exception e ){}
}
```

Download the Code!
www.JavaDevelopersJournal.com

SYS-CON MEDIA

PUBLISHER, PRESIDENT, AND CEO
FUAT A. KIRCAALI fuat@sys-con.com
CHIEF OPERATING OFFICER
MARK HARABEDIAN mark@sys-con.com
VICE PRESIDENT, BUSINESS DEVELOPMENT
GRISHA DAVIDA grisha@sys-con.com

ADVERTISING
SENIOR VICE PRESIDENT, SALES AND MARKETING
CARMEN GONZALEZ carmen@sys-con.com
VICE PRESIDENT, SALES AND MARKETING
MILES SILVERMAN miles@sys-con.com
ADVERTISING SALES DIRECTOR
ROBYN FORMA roybn@sys-con.com
ADVERTISING ACCOUNT MANAGER
MEGAN RING megan@sys-con.com
ASSOCIATE SALES MANAGERS
CARRIE GEBERT carrieg@sys-con.com
KRISTIN KUHNLE kristen@sys-con.com
ALISA CATALANO alisa@sys-con.com
LEAH HITTMAN leah@sys-con.com

EDITORIAL
EXECUTIVE EDITOR
NANCY VALENTINE nancy@sys-con.com
EDITOR
M'LOU PINKHAM mpinkham@sys-con.com
MANAGING EDITOR
CHERYL VAN SISE cheryl@sys-con.com
ASSOCIATE EDITORS
JAMIE MATUSOW jamie@sys-con.com
GAIL SCHULTZ gail@sys-con.com
JEAN CASSIDY jean@sys-con.com
ASSISTANT EDITOR
JENNIFER STILLEY jennifier@sys-con.com
ONLINE EDITOR
LIN GOETZ lin@sys-con.com

PRODUCTION
VICE PRESIDENT, PRODUCTION AND DESIGN
JIM MORGAN jim@sys-con.com
LEAD DESIGNER
LOUIS F. CUFFARI louis@sys-con.com
ART DIRECTOR
ALEX BOTERO alex@sys-con.com
ASSOCIATE ART DIRECTORS
CATHRYN BURAK cathyb@sys-con.com
RICHARD SILVERBERG richards@sys-con.com
AARATHI VENKATARAMAN aarathi@sys-con.com
ASSISTANT ART DIRECTOR
TAMI BEATTY tami@sys-con.com

WEB SERVICES
WEBMASTER
ROBERT DIAMOND robert@sys-con.com
WEB DESIGNERS
STEPHEN KILMURRAY stephen@sys-con.com
CHRISTOPHER CROCE chris@sys-con.com
CATALIN STANCESCU catalin@sys-con.com

ACCOUNTING
CHIEF FINANCIAL OFFICER
BRUCE KANNER bruce@sys-con.com
ASSISTANT CONTROLLER
JUDITH CALNAN judith@sys-con.com
ACCOUNTS RECEIVABLE
JAN BRAIDECH jan@sys-con.com
ACCOUNTS PAYABLE
JOAN LAROSE joan@sys-con.com
ACCOUNTING CLERK
BETTY WHITE betty@sys-con.com

SYS-CON EVENTS
VICE PRESIDENT, SYS-CON EVENTS
CATHY WALTERS cathyw@sys-con.com
CONFERENCE MANAGER
MICHAEL LYNCH mike@sys-con.com
SALES EXECUTIVES, EXHIBITS
MICHAEL PESICK michael@sys-con.com
RICHARD ANDERSON richard@sys-con.com

CUSTOMER RELATIONS/JDJ STORE
MANAGER, CUSTOMER RELATIONS/JDJ STORE
ANTHONY D. SPITZER tony@sys-con.com

Simplex Knowledge Company

skc.com

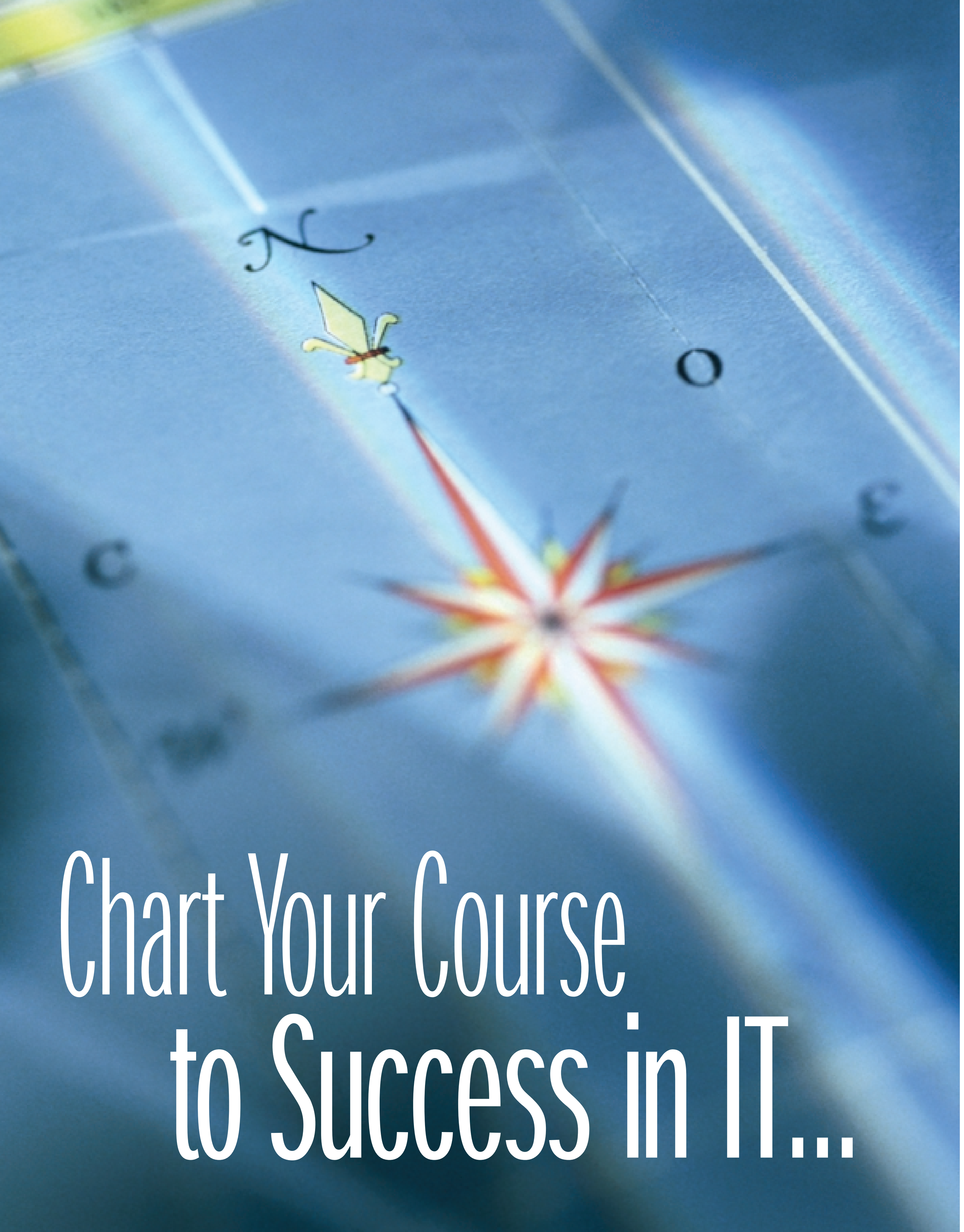


Chart Your Course to Success in IT...

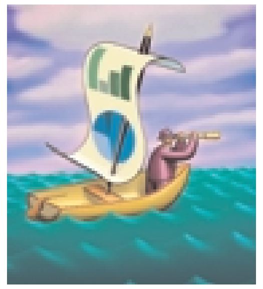
...order your copy of

Java Trends: 2003

Available July 1*

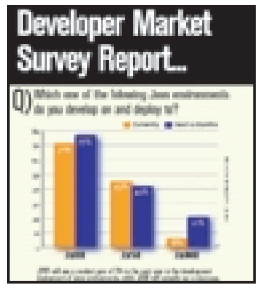
EnginData's research is invaluable to leading IT vendors, integrators, Fortune 500 companies, trade-show organizers, publishers, and e-business organizations worldwide.

Don't go astray. In the vast sea of Internet technology, market conditions change constantly. Will Java remain the hot platform it is today? Will C# rapidly gain ground? What are the world's foremost Java developers aiming their sights toward? Which companies come to mind when planning their next project? How can their thinking direct your efforts?



EnginData Research has studied the IT industry extensively, spotting many key trends and alerting industry leaders along the way. In the first quarter of 2002, they embarked on their most challenging mission ever: the most in-depth study ever done of the Java development marketplace.

After months of analyzing and cross-referencing more than 10,500 comprehensive questionnaires, the results are now available.



Here's just a sample of the critical data points the Java report delivers...

- ✔ Current and projected usage of languages and platforms
- ✔ Multiple rankings of hundreds of software vendors and tools
- ✔ Types of applications being developed
- ✔ Databases being used
- ✔ Purchasing patterns
- ✔ Company size
- ✔ Development and purchasing timelines
- ✔ Perceptions and usage of Web services
- ✔ Preferred Java IDE
- ✔ J2EE, J2ME, J2SE usage comparisons

As an IT specialist, **EnginData Research** focuses exclusively on three leading drivers in IT today – Java, Web services, and wireless development. Coming soon, our Web services survey will deliver such critical knowledge as:

- ✔ Time frame for developing Web services – enabled apps
- ✔ Percentage of apps with Web services today
- ✔ Sourcing and hosting Web services
- ✔ Perceived leaders in Web services tools and solutions

Navigate your company through the challenging IT marketplace with the trusted knowledge and intelligence of **EnginData Research**. Order your copy of the 2002–2003 Java market study by calling Margie Downs at 201-802-3082, or visiting our Web site.

engindata RESEARCH

www.engindata.com

*A limited number of preview copies will be available at our booth (#624) at JDJEdge International Java Developer's Conference & Expo, June 24–27, at the Jacob Javits Convention Center in New York.

SAVE 30% off the annual newsstand rate

JAVA DEVELOPER'S JOURNAL

Offer subject to change without notice

ANNUAL NEWSSTAND RATE
~~$71.88~~
YOU PAY
$49.99
YOU SAVE
30% Off the Newsstand Rate

DON'T MISS AN ISSUE!

Receive **12** issues of ***Java Developer's Journal*** for only **$49.99**! That's a savings of **$21.89** off the cover price. Sign up online at **www.sys-con.com** or call **1 800 513-7111** and subscribe today!

Here's what you'll find in every issue of *JDJ*:

- Exclusive coverage **J2EE J2SE J2ME**
- Exclusive feature articles
- Interviews with the hottest names in Java
- Latest Java product reviews
- Industry watch

Java Design

Creating flexible code

WRITTEN BY MICHAEL BARLOTTA

Java classes should be designed to enhance their reusability and flexibility. Coding to an object type rather than an implementation by using interfaces or abstract classes can help us achieve both flexibility and reusability.

What Is an Abstract Class?

Everything in Java is an object, so to write Java programs we need to define classes. A Java class in turn defines the type and the available methods for that type. It also provides the implementation of the methods that it defines. This kind of class is called a concrete class. An abstract class, defined in section 8.1.1.1 of the Java Language Specification (JLS), defines the type and the available methods but is not required to implement any of the methods that it defines.

The following code shows a simple abstract class named Base.

```
public abstract class Base {
 public abstract String f1();
 public String f2(String s) {
   return "Base.fs("+s+")";
 }
}
```

An abstract class is denoted by the keyword abstract in the class declaration and usually contains one or more abstract methods. An abstract method declaration defines the method name, number and type of parameters, return type, and throws clause, but does not have an implementation. In the declaration an abstract method is denoted by the abstract keyword, as is the case with method f1 in the class Base defined earlier.

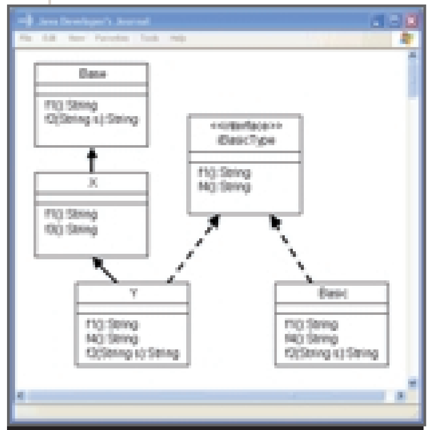


FIGURE 1 Relationships of simple objects in this article

Why design with an abstract class instead of a concrete class? Using an abstract class allows you to define the interface of a type and still provide method implementations without having to implement the entire set of methods. The term *interface* is used here to mean the publicly accessible functions on the class, not the Java interface, which we'll look at next. The abstract class represents a type at a very generic level. Although operations can be defined for the type, the abstract class can't provide all the functionality without becoming too specific.

An abstract class is incomplete and relies on its subclasses to complete the implementation. In the following code sample the class *X* inherits from the class Base by extending it. The class Base is considered the superclass of class *X*. The class *X* is considered the subclass (or subtype) of class Base. For *X* to avoid being declared abstract, it must implement the abstract method f1 defined in the class Base. The definition of class *X* is listed below:

```
public class X extends Base{

 public String f1() {
  return "X.f1()";
 }
```

SYS-CON Media

www.sys-con.com

CHECK OUT THE NEW XML-J!

XML JOURNAL

Offer subject to change without notice

ANNUAL NEWSSTAND RATE
~~$83.88~~
YOU PAY
$77.99
YOU SAVE
$5.89 Off the Newsstand Rate

DON'T MISS AN ISSUE!
Receive 12 issues of *XML-Journal* for only $77.99! That's a savings of $5.89 off the annual newsstand rate.
Sign up online at www.sys-con.com or call 1 800 513-7111 and subscribe today!

Wait till you see June...
ENHANCED industry and technical coverage
Expanded BUSINESS focus
Dynamic new EDITORIAL TEAM
SECTION-SPECIFIC COVERAGE:

CONTENT MANAGEMENT
Document XSLT – Automatically
How to document conditional text processing for business users
by Karl B. Schwamb & Kenneth J. Hughes

DATA MANAGEMENT
Managing Data Sources with XSLT
An approach to transform any type of data source to XML
by Craig King

ENTERPRISE SOLUTIONS
UBL and Web Services
The gaps in Web services, and the steps to resolve them, need to be assessed
by Matthew Gertner

XML LABS
Desperately Seeking... Help for XML Schema
Are the complexities...too complex?
by Tom Gaven

THE ULTIMATE XML ENTERPRISE RESOURCE

```
 public String f3() {
  return "X.f3()";
 }
}
```

Some important points to remember about abstract classes:

- An abstract class cannot be instantiated.
- An abstract class can implement zero, one, or more methods.
- A class with an abstract method must be declared abstract.
- A class can be declared abstract even if it has no abstract methods.
- Methods that are static, private, and/or final cannot be abstract.

What Is an Interface?

A Java interface defines a type and the available methods for that type. It does not, and, in fact, cannot provide an implementation for any of the methods it defines. An interface is defined in section 9 of the Java Language Specification.

The following is a simple interface, iBasicType:

```
public interface iBasicType {
 public String f1();
 public String f4();
}
```

The interface and all the methods on it are automatically considered abstract and do not require the abstract keyword.

To be of any use a Java interface must be implemented by a Java class. In the following code sample, the class Basic implements the interface iBasicType. The class Basic must provide an implementation for every method in the interface, otherwise it must be declared an abstract class.

```
public class Basic implements iBasicType {

 public String f1() {
  return "Basic.f1()";
 }

 public String f4() {
  return "Basic.f4()";
 }

public String f2(String s) {
return "Basic.fs("+s+")";
}
}
```

A class implementing an interface can also declare additional methods that aren't part of the interface. This is shown in the Basic class definition by defining method f2, which isn't part of the iBasicType interface.

Some important points to remember:

- An interface cannot be instantiated.
- An interface has no implementation.
- All the methods on an interface are considered public and abstract.
- All the fields on an interface are considered public, static, and final.
- An interface can extend other interfaces.
- A class can implement one or more interfaces.

The Power of Dealing with Types

Java is a strongly typed language, so every object in Java has a type. The type associated with any particular object is its class name. For example, when the class Customer is instantiated, the object is said to be of the type Customer. Each instance of an object may have a different state, but all objects of the same type have the same set of methods because they're created from the same class. Abstract classes and interfaces give us more flexibility in our programs because they allow us to

BATTLE IT OUT!
Compete against your fellow developers, the clock and our host in a Java™ technology trivia contest to see who is the "strongest geek!"

CO-SPONSORED BY
Rational the software development company
IBM

Let's Play! WEAKEST GEEK

GRAND PRIZE WINNER TO CHOOSE FROM...
1 2 3

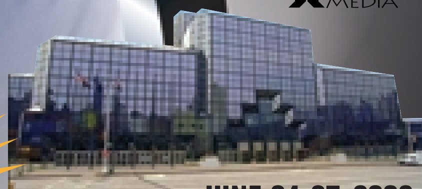
ALSO... 6 WINNERS WILL RECIEVE A COPY OF RATIONAL XDE PROFESSIONAL, AND IBM STUDIO APPLICATION DEVELOPER

PRESENTED BY JAVA DEVELOPERS JOURNAL

Hosted by JDJ Editor-in-Chief Alan Williamson

OWNED AND PRODUCED BY SYS-CON EVENTS SYS-CON MEDIA

$10K CASH!
BRAND NEW 2002 HARLEY DAVIDSON "SPORTSTER" MOTORCYLE!
BRAND NEW 2002 BMW "MINI COOPER" 2YR LEASE!



JUNE 24-27, 2002
JACOB JAVITS CONVENTION CENTER • NYC
PLAY THE GAME AT THE WORLD'S LARGEST IT CONFERENCE AND EXPO AT JACOB JAVITS IN NEW YORK CITY
web services EDGE conference & expo | JDJ EDGE conference & expo | XML EDGE conference & expo

SAVE 30% off the annual newsstand rate

wireless BUSINESS & TECHNOLOGY

Offer subject to change without notice

ANNUAL NEWSSTAND RATE ~~$71.88~~
YOU PAY $49.99
YOU SAVE 30% Off the Newsstand Rate

DON'T MISS AN ISSUE!

Receive 12 issues of *Wireless Business & Technology* for only $49.99! That's a savings of 30% off the cover price. Sign up online at www.sys-con.com or call 1 800 513-7111 and subscribe today!

In June *WBT*:

Tech Update: What's Hot in Wireless
We can look forward to tangible, near-term gains that will change how and where we use wireless technologies in 10 key areas.

Retailers Check Out Portable Point of Sale Systems
Wireless terminals such as PalmPilots can emulate the cash register and enable credit card customers to skip the lines.

The Wizard of Wireless
Applications such as voice-activated dialing, voice portals, and handsfree conferencing may help the wireless industry in its quest for killer apps.

Mobile Browser Roundup
If your IT department is planning to offer remote users enterprise access via a mobile browser, you need to know what's available and the unique characteristics they offer.

look at an object as a different type instead of just using its class name.

For example, the object *Y*, shown in Figure 1, can be created from the following line of code:

```
Y myYObj = new Y();
```

The variable myYObj contains an object reference of type *Y* that points to a *Y* object in memory after this code is executed. Through the *Y* object reference all the methods defined on the *Y* object and any class it extends can be invoked. Since *Y* extends the class *X*, they're related and we can treat the subclass *Y* as if it were actually any one of its superclasses, in this case *X* or Base; e.g., the following code is completely valid:

```
X myXObj = myYObj;
```

The variable myXObj contains an object reference of type *X*, but the reference points to a *Y* object in memory after this line of code is executed. This allows us to look at the *Y* object as if it were really of type *X*. The compiler can safely place references to subclasses into a variable that's declared to be a superclass of that subclass, since the subclass is guaranteed to contain all the methods and variables that can be called on the superclass. Simply, *Y* extends *X*, so the compiler is assured that any method that can be called on *X* can also be called on *Y*. However, the reverse is not guaranteed to be true so the following code is not valid:

```
Y anotherYObj = new X();             // This is not valid
```

Dealing with a subclass object as if it were really one of its superclasses is called upcasting or narrowing. It's important to understand that the object reference, which points to an object in memory, does not determine the type of the object that gets the request to invoke a method, nor does it change the type of the object that was instantiated. The object reference does, however, affect how the object it points to can be operated on. The object reference defines the type through which the calling program sees the object in memory. You can think of the *X* object reference as a filter that allows only calls to methods that are defined in the *X* class and any class it extends. Looking at the class diagram in Figure 1, we can call the f1, f2, and f3 methods through the *X* object reference. But we couldn't call the f4 method even though this is a valid method on the *Y* object, because it's not defined by the class *X* or any of its superclasses.

When we call methods through the *X* object reference using the myXObj variable, we're actually invoking methods in the object that the reference points to in memory. In this case it's a *Y* object instance.

```
myXObj.f1();    // returns "Y.f1()"
myXObj.f3();    // returns "X.f3()"
```

These method calls give the results shown in the comment sections based on the following definition of the *Y* class:

```
public class Y extends X implements iBasicType {

 public String f1() {
  return "Y.f1()";
 }

 public String f2(String s) {
  return "Y.fs("+s+")";
 }

 public String f4() {
  return "Y.f4()";
 }
}
```

WebServices JOURNAL .NET J2EE XML — XML JOURNAL

THE FIRST & ONLY WEB SERVICES RESOURCE CD!

WEB SERVICES RESOURCE CD

THE SECRETS OF THE WEB SERVICES MASTERS

INCLUDES EXCLUSIVE .NET ARTICLES

$79 + S&H

Special Limited time Price

$119 CD VALUE FROM WEB SERVICES JOURNAL

Now Shipping

web services EDGE conference & expo $100 COUPON INSIDE

MORE THAN 400 EXCLUSIVE WEB SERVICES & XML ARTICLES

EDITED BY SEAN RHODY

3 YEARS 25 ISSUES 400 ARTICLES ONE CD

THE MOST COMPLETE LIBRARY OF EXCLUSIVE WSJ & XML-J ARTICLES ON ONE CD!

"The Secrets of the Web Services Masters"
CD is edited by well-known WSJ Editor-in-Chief Sean Rhody and organized into more than 40 chapters containing more than 400 exclusive WSJ & XML-J articles.

every issue of wsj & xml-j ever published

www.JDJSTORE.com

ONLINE ORDER AT JDJSTORE.COM SAVE $40

OFFER EXPIRES JUNE 30, 2002

SUBSCRIBE AND SAVE

WebServices JOURNAL

Offer subject to change without notice

ANNUAL NEWSSTAND RATE
~~$83.88~~
YOU PAY
$69.99
YOU SAVE
$13.89 Off the Newsstand Rate

DON'T MISS AN ISSUE!

Receive 12 issues of *Web Services Journal* for only $69.99! That's a savings of $13.89 off the annual newsstand rate. Sign up online at www.sys-con.com or call 1 800 513-7111 and subscribe today!

In June *WSJ:*

Business Process Management and Web Services
Design deploy, execute, analyze, and optimize business processes as Web services

Making Business Processes Manageable
The need to take advantage of explicitly defined processes

Business Process Management: Tools for Eliminating Waste and Adding Value
Making a technology investment

Web Services and Business Process Management
Start thinking in terms of the process – not the application

Accelerating the Adoption of Web Services in Your Enterprise
Think ahead and you can avoid the pitfalls

Notice that the implementation of the f1 method on the *Y* class is run even though we're using an *X* object reference. In this example, the *Y* class does not define its own f3 method, so the implementation of the f3 method on the *X* class is executed.

It's possible to point an object reference to an object that is its subclass because Java performs dynamic binding. Objects communicate by sending messages to each other through references. This is basically a means of calling methods on the object. Dynamic binding allows messages to be sent to an object without knowing the exact type of the object that the message will be sent to when the code is written. When the code is executed and a method is invoked, the JVM determines which method is actually executed based on which object is being pointed to by the object reference at runtime.

The message to execute a particular method cannot be sent to just any object. Java is a strongly typed language so the compiler makes sure the assignment is type-safe. A variable has a type that never changes, so the myXObj variable will always hold an *X* object reference. When the variable type is a reference type (arrays, classes, and interfaces), it can point to different objects that are related via inheritance or an interface. Specifically, a reference variable can be assigned a reference to any object whose class is:

- Of the same type as that of the reference variable declaration
- A subtype of the type that the reference variable is declared to be

Or any object that implements:

- The same interface that the reference variable is declared to be
- An interface that is a subtype of the interface that the reference variable is declared to be

An Example with an Interface

We can declare variables to have a type that is defined as an abstract class or an interface even though they can't be instantiated. Given the class hierarchy shown in Figure 1, a variable can be declared as follows:

```
iBasicType myBasicObj;
```

The variable, myBasicObj, holds an object reference to the interface iBasicType. We can instantiate objects that implement the interface using the code sample below:

```
iBasicType myBasicObj = new Basic();
```

The variable, myBasicObj, holds an iBasicType reference that points to a Basic object in memory. Since the class *Y* also implements the interface iBasicType, the following code is also valid:

```
iBasicType anotherBasicObj = new Y();
```

By implementing an interface, the object is also declaring that it is of that type. For example, the class *Y* can be said to be of type *Y*, its class name, as well as of type iBasicType, since it implements that interface.

Like the variable that holds an object reference to a class type, an object reference to an interface type defines the way the calling program sees the object in memory. So we can call the f1 and f4 methods through the iBasicType object reference. But we could not call the f2 method even though this is a valid method on both the Basic and the *Y* object, because it's not defined by the interface (see Figure 1).

Wrapping It Up

Hopefully, you'll begin to see the potential flexibility we can design into our applications by coding to different types. Applying this in a practical sense will be the focus of our next article, until then, happy coding.

Author Bio
Michael Barlotta is the director of technology at AEGIS.net Inc (www.AEGIS.net).

mike.barlotta@aegis.net

THE INSIDER INTELLIGENCE YOU NEED...

TO KEEP AHEAD OF THE CURVE

Go to www.SYS-CON.com

JAVA Industry > Newsletter
WebServices Industry > Newsletter
XML JOURNAL Industry > Newsletter
wireless Industry > Newsletter
WebLogic Industry > Newsletter
WebSphere Industry > Newsletter
COLDFUSION Industry > Newsletter
LINUX WEEK Industry > Newsletter

SELECT THE INDUSTRY NEWSLETTERS THAT MATCH YOUR NEEDS! CHOOSE ONE OR TRY THEM ALL!

FREE E-Newsletters SIGN UP TODAY!

The most innovative products, new releases, interviews, industry developments, and plenty of solid *i*-technology news can be found in SYS-CON Media's Industry Newsletters. Targeted to meet your professional needs, each e-mail is informative, insightful, and to the point. They're free, and your subscription is just a mouse-click away at www.sys-con.com.

Exclusively from the World's Leading *i*-Technology Publisher SYS-CON MEDIA

WebServices JOURNAL .NET J2EE XML

Only $69.99 for 1 year (12 issues)
Newsstand price $83.88 for 1 year

Special Introductory Offer SAVE $13.89*

The world's leading independent Web Services information resource

Get Up to Speed with the Fourth Wave in Software Development

- Real-World Web Services: Is It Really XML's Killer App?
- Demystifying ebXML: A Plain-English Introduction
- Authentication, Authorization, and Auditing: Securing Web Services
- Wireless: Enable Your WAP Projects for Web Services
- The Web Services Marketplace: An Overview of Tools, Engines, and Servers
- Building Wireless Applications with Web Services
- How to Develop and Market Your Web Services
- Integrating XML in a Web Services Environment
- Real-World UDDI
- WSDL: Definitions and Network Endpoints
- Implementing SOAP-Compliant Apps
- Deploying EJBs in a Web Services Environment
- Swing-Compliant Web Services
- and much, much more!

www.wsj2.com

*OFFER SUBJECT TO CHANGE WITHOUT NOTICE

SYS-CON MEDIA

SPECIAL REPORT: *JAVA DEVELOPER'S JOURNAL* APRIL EDITORIAL "THERE MAY BE TROUBLE AHEAD FOR JAVA..." RALLIES JAVA CAMP

by Jeremy Geelan

ALAN WILLIAMSON'S APRIL EDITORIAL was read by more than 100,000 people within two hours of its posting at www.sys-con.com/java. Over 500 readers responded within hours of its publication. The editorial was instantly picked up and simultaneously published at the Slashdot.org and JavaLobby.org Web sites.

The Great Java Debate has been raging as a direct result of the technical points raised by Alan Williamson, ***Java Developer's Journal*** editor-in-chief, in his editorial (Vol. 7, issue 4) about C#, Microsoft's rival language to Java.

Everyone had been talking about C#, Williamson felt, and in true Microsoft "Chinese whispers" style, it spread rapidly. "We felt it was time for *JDJ* to find out the real truth behind this new language and present the facts as we found them," he says.

There will always be particularly virulent controversy whenever an industry commentator mentions Java and Microsoft Corp. in the same breath. Comments posted to the *Java Developer's Journal* Web site (www.sys-con.com/java/article.cfm?id=1401) range from praise such as this from business consultant David Bolsover (david@bolsover.com) who writes, "I have to agree with much of what Alan wrote in his editorial – the level of ignorance and misunderstanding among so-called IT professionals is astounding" to dramatic criticism such as this from Canadian software developer Christian Ouellet (christian099@sympatico.ca), who strongly believes that such an editorial "is not what the Java community needs" to the extent that, as he says: "that's why I will burn all my *JDJ* issues…"

Well, Java always did evoke strong emotions!

"*Java Developer's Journal* is proud to take the lead in peering over the horizon; that's what readers (developers) expect from us," says editor-in-chief Williamson. "That makes me kind of chief scout on behalf of the readership, heading up the gullies and reporting back on what I find."

"The Java community is one of the most passionate and open set of people I have seen in my life," he says, "and this strength will ensure that Java will be a formidable force against the alternatives that come up on the radar, such as Microsoft's C#."

The feedback generated falls into two camps: those who feel the writing is on the wall for Java, and those who believe that Java has never been stronger and that never before have we seen so many applications and general forward movement in the Java space as we see today.

Sun Needs to Wake Up and Smell the Java

"Alan Williamson is right and many are too blind to see," writes Steven Tower (tower_@hotmail.com). "I've been a Java developer for almost seven years now," he continues. "I was very lucky to have gotten in on the ground floor. However, I have discussed lately with other developers that we really are at a crossroads. Sun has been arrogant to the point of insult and has shrugged off real concerns and problems. Sun needs to completely embrace their community, open source and otherwise. They need to wake up and smell the Java, so to speak."

Tower explains: "I hope and expect Java to be my language of choice for years to come. But I can't deny a viable alternative. Sun has completely failed on the desktop. Yeah, you can argue, oh look at this great app and that one, but truth be told Sun blew it; OS/2 had some great apps too. If C# makes even small inroads on the server and on the desktop, Java is in real trouble. Microsoft doesn't need massive wins; once they have their foot in the door on both, they will start to squeeze on all sides. Don't hide your heads in the sand or hold them so high in arrogance in the belief that because they haven't already, Microsoft won't find a way."

Developers like to think about the look and feel of an app they plan to develop, and they like to think about the algorithms that will make it run efficiently. If Java happens to be the best language for the job, then that's what they'll use. But they're not tied to any language, and this is why – as Williamson was trying to point out – it's important for Javaland to consider the potential rise of C# very carefully.

Java Will Benefit from Some Healthy Competition

"He's right on the money!" says Michael Julson (mjulson@visualfire.com). "I agree with Alan's suggestions and beliefs of where Java might be in the future. Sun has been busy fracturing the language into so many pieces with the hope that it will be the language for all situations and all needs. Because of this fracturing, it's become a weaker language and has suffered."

"I welcome C#," Julson continues. "With its similarities to Java, it puts some tougher competition to Sun, IBM, et al, and will make the language better off for it. Beyond that, I'll be able to use it for desktop apps that don't crawl at a snail's pace."

David Bolsover, mentioned earlier, echoed this theme of the responsibility that Java supporters have to spread the word. "The Java community must make renewed efforts to communicate the merits of Java to the wider community if it is to survive," he says. "Personally, I think Sun must take the lead in this – a few well-placed full page ads in national daily papers wouldn't go amiss, free CDs on magazine covers – anything that promotes the Java message."

And Aisha Fenton (aishafenton@mac.com) too feels that Williamson is sounding a very timely rallying cry: "C# is set to become the dominant client-side language," she explains. "MS will make it very fast and a natural choice to program in for Windows (even games could be easily written in it, since they'll have good Direct X support). We can't just ignore it and attack anyone that tries to talk about it. We have five years to make Java better, let's get going."

Nobody Knows What's Going to Happen

"It's just branding. Get over it," advises Jason Norman (jason@elluzion.net), a health systems software engineer at Vanderbilt University Medical Center in Nashville, Tennessee.

"Let's face it," Norman says, "nobody knows what's going to happen. On the one hand we have MS. They have a pretty strong following in the business world. Their branding is fantastic. They have all sorts of cool commercials (however cryptic and uninformative they may be) and for some strange reason, lots of people believe in them."

"On the other hand," he continues, "we have Java, billions of lines of Java code floating around the Internet. There are lots of enterprise applications and systems in place, millions and millions of dollars already invested. Who will want to basically flush away everything they already have just to jump on MS's latest buzzword?"

"Just calm down," adds Norman. "In five years, neither Java nor .NET (if it is still around) will be anything like they are now. This whole debate will seem silly and pointless."

Critically Important Commentary or Soul Selling?

Miles Parker (milesparker@earthlink.net) is in no doubt: "What an incredibly important commentary," he declares. "It is literally laughable to suggest that Alan has some kind of pro-MS, anti-Java bent. Instead, what he's offering is something we all need to hear."

"Strangely," Parker continues, "companies and developer communities have never lost by overestimating Microsoft; they have always lost by underestimating them!…Please, please, let us not be like Netscape ("we own the browser market"), IBM, Sybase, (soon to be Palm), and so many companies in between."

Referring to some of the criticisms that the editorial provoked, Parker comments: "That Alan has received such a load of BS because he is willing to ask real questions is dismaying, to say the least, and makes me think that Java developers are more concerned about living in a state of comfortable and smug denial than fighting to protect the diversity and strength that Java and associated tools have provided."

"If we aren't honest with ourselves and willing to fight this battle day to day in the trenches," he continues, "we will lose, and indeed there will not be much left of Java in five years. Ask yourself, what did you think of the prospects of Netscape in 1997? Now we know that they were already doomed, due in large part to their own arrogance. We can't afford to have our heads in the sand! .NET is a very real threat."

Christophe de Dinechin (descubes@earthlink.net) believes no computer language can expect immortality, Java included. "Yes, Java is going to die," he says, "and five years might be a good time frame. But C# is not the reason. The reason is that we will need to do things in five years that Java doesn't do well. A new environment will replace Java, just as Java replaced C++ when the Internet became the place to be, just as C++ replaced C, just as C replaced Pascal, etc. New tasks=new programming language."

"By this reasoning, however," de Dinechin adds, "C# should not displace Java, simply because it only does what Java does, with a little more. Naturally, C# has the capability to evolve."

Avram Aelony (aelony@mpi.com) disagrees that Java has only five years left. "There seem to be quite a few elderly languages with less promise than Java," he points out, "that refuse to die. Diversity is good. If you could write an app in your favorite idiomatic language and the compiler's task was to make it run fast anywhere, then there's also room for C# in the world."

For Sarwar Mansoor (smansoor@sbcglobal.net), however, C# is simply not something to be mentioned in polite company. "After reading this I have lost faith in *JDJ*," he complains. "I thought this was a true journal on Java. I am reading *Java Pro* from now on. I code for a living and love coding, but I am not selling my soul."

John Harrisburg (jharrisburg@adt.com) takes issue with this. "You can read *JavaPro* until they go out of business just like *JavaReport* did," he retorts. "Do you remember *Java Report*? Where are they now? Gone.…*JDJ* is the only honest Java magazine out there."

Tools Are the Key

British developer Nick Riordan (nick@anamatica.com) returns the discussion to a more technical level. "I spent 10 years building apps exclusively for the Microsoft platform using Microsoft tools (C and C++)," he explains. "Two years ago I changed jobs and have since been working with Java/CORBA and EJB. I like Java, but I think C# offers similar benefits. It has already been said in this discussion that the client is important, and let's face it, Swing is probably the weakest area of Java."

"But my real issue," Riordan goes on, "is the lack of proper tools for Java. If you've spent any time working with MS technologies, you really appreciate the properly integrated, high-performance, polished feel – it makes development a pleasure. Recently I moved across to IDEA as my IDE in Java – it's the best Java IDE I've found so far (I want to like NetBeans, but it never seems to be finished and the performance stinks). IDEA is about as good as Visual C++ 4.2 – a product that shipped five years ago."

"I want seamless end-to-end debugging (client, middle tier, SQL)," says Riordan. "I want high speed – and no performance degradation when running under the debugger. I want folding editors, proper dialog editing, etc., etc. This is the point: you can be so much more productive under C# just because of the tools."

Is a "Religious" Following a Symptom of Decline?

What does this whole passionate discussion of the future of Java mean? A very thought-provoking perspective comes from Mark Miller (mmille10@attbi.com), who sees what he claims are signs of an oft-repeated cycle of behavior.

"I've seen this happen too often in the technological world," he says, "and I've been in it since the early 1980s. It's a pattern. A technology starts faltering, and the ways part. Some people use it until they feel its usefulness is exhausted and then move on to something else. The others form a cultlike following that is dead set against using any other technology aside from the one they love."

It's this cultlike behavior that Miller claims to discern in the current behavior of what he sees as Javaland's die-hards and in their reactions to Alan Williamson's editorial.

"They say things like, 'If only they would do X, then people would see how great it is and start using it'…They don't look at integrating other useful technologies, because, of course, theirs is so wonderful. They develop strong biases against outside technologies, in fact. They don't see their own technology's weaknesses; they don't want to see them."

"Don't make this a religious or social movement," he concludes, "Make it a wake-up call to Sun that they need to fundamentally change the way they approach Java's development."

"The feedback generated falls into two camps: those who feel the writing is on the wall for Java, and those who believe that Java has never been stronger"

What do you think? What is the future of Java? What should be done to boost Java? Can Java and C# coexist peacefully? To add your comments, go to www.sys-con.com/java/article.cfm?id=1401.

FREE TUTORIAL JUNE 27TH WITH WEB SERVICES EDGE 2002 EAST REGISTRATION. LIMITED OFFER!

TO REGISTER: www.sys-con.com or Call 201 802-3069

web services EDGE world tour 2002

$195 REGISTRATION FOR SYS-CON SUBSCRIBERS
BEST EDUCATIONAL VALUE FOR THE HOTTEST TECHNOLOGY SKILLS!

Take Your Career to the Next Level!

SEATING IS LIMITED. REGISTER NOW FOR THE CITY NEAREST YOU! WWW.SYS-CON.COM

Learn How to Create, Test and Deploy Enterprise-Class Web Services Applications

TAUGHT BY THE INNOVATORS AND THOUGHT LEADERS IN WEB SERVICES

EXPERT PRACTITIONERS TAKING AN APPLIED APPROACH WILL PRESENT TOPICS INCLUDING BASIC TECHNOLOGIES SUCH AS SOAP, WSDL, UDDI AND XML, PLUS MORE ADVANCED ISSUES SUCH AS SECURITY, EXPOSING LEGACY SYSTEMS AND REMOTE REFERENCES.

RECEIVE 4 FREE 1-YEAR SUBSCRIPTIONS $345 VALUE!

SHARPEN YOUR PROFESSIONAL SKILLS.

KEEP UP WITH THE TECHNOLOGY EVOLUTION!

"Presented an excellent overview of Web services. Great inside knowledge of both the new and old protocols. Great insight into the code piece."

– Rodrigo Frontecilla

"Very articulate on the Web services SOAP topic and well-prepared for many questions. I've learned a lot from this seminar and I appreciate this seminar for my job. Thank you!"

– Kenneth Unpingco, Southern Wine & Spirits of America

"I liked the overview of Web services and the use of specific tools to display ways to distribute Web services. Good for getting up to speed on the concepts."

– B. Ashton, Stopjetlag.com

Echoed over and over by Web Services Edge World Tour Attendees:

"Good balance of theory and demonstration."

"Excellent scope and depth for my background at this time. Use of examples was good."

"It was tailored toward my needs as a novice to SOAP Web services – and they explained everything."

WHO SHOULD ATTEND:

- **Architects**
- **Developers**
- **Programmers**
- **IS/IT Managers**
- **C-Level Executives**
- ***i*-Technology Professionals**

IF YOU MISSED THESE...

BOSTON, MA (Boston Marriott Newton) SOLD OUT!
WASHINGTON, DC (Tysons Corner Marriott) SOLD OUT!
NEW YORK, NY (Doubletree Guest Suites) SOLD OUT!
SAN FRANCISCO, CA (Marriott San Francisco) SOLD OUT! CLASSES ADDED

BE SURE NOT TO MISS THESE...

Each city will be sponsored by a leading Web services company

...COMING TO A CITY NEAR YOU

2002	
NEW YORK AT WEB SERVICES EDGE 2002 EAST	JUNE 27
BOSTON	JULY 10
SAN FRANCISCO	AUGUST 6
SEATTLE	AUGUST 27
AUSTIN	SEPTEMBER 10
LOS ANGELES	SEPTEMBER 19
SAN JOSE AT WEB SERVICES EDGE 2002 WEST	OCTOBER 3
CHICAGO	OCTOBER 17
ATLANTA	OCTOBER 29
MINNEAPOLIS	NOVEMBER 7
NEW YORK	NOVEMBER 18
SAN FRANCISCO	DECEMBER 3
2003	
CHARLOTTE	JANUARY 7
MIAMI	JANUARY 14
DALLAS	FEBRUARY 4
BALTIMORE	FEBRUARY 20
BOSTON	MARCH 11

REGISTER WITH A COLLEAGUE AND SAVE 15% OFF THE LOWEST REGISTRATION FEE.

TOPICS HAVE INCLUDED:
Developing SOAP Web Services
Architecting J2EE Web Services

On May 13th, the San Francisco tutorial drew a record 601 registrations.

SYS-CON *i*-TECHNOLOGY EDUCATION

REGISTRATION FOR EACH CITY CLOSES THREE BUSINESS DAYS BEFORE EACH TUTORIAL DATE. DON'T DELAY. SEATING IS LIMITED.
NON-SUBSCRIBERS: REGISTER FOR $245 AND RECEIVE THREE FREE ONE-YEAR SUBSCRIPTIONS TO *WEB SERVICES JOURNAL, JAVA DEVELOPER'S JOURNAL*, AND *XML-JOURNAL*, PLUS YOUR CHOICE OF *BEA WEBLOGIC DEVELOPER'S JOURNAL* OR *WEBSPHERE DEVELOPER'S JOURNAL*, A $345 VALUE!

TO REGISTER: www.sys-con.com or Call 201 802-3069

The C (VS.NET) Community

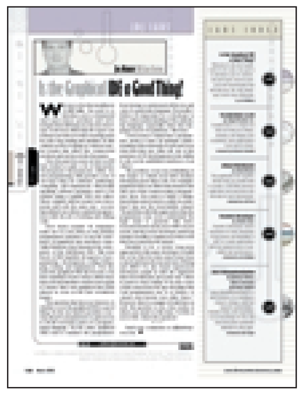
As far as I can tell, many developers think Java is dead because "VS.NET is such a great IDE" and "C# is such a better language" ("In Search of The Community" by Keith Brown [Vol. 7, issue 4]). The problem is that they don't understand why we use Java. C# is a good language and VS.NET is a great IDE, but there are some things I don't like about it and I don't think it's any better than Java and my favorite IDE.

Also, only VS.NET is .NET. Everything else is still COM, etc. Which means in all reality it will be quite a while before everything MS is .NET-ed (MSMQ, MTS, SQL Server, Exchange, etc.). So for those hoping or saying that .NET/C# will soon be available on Linux – well, if it's taking MS this long with all their resources, how long do you think it will take less-funded and less-supported projects such as Mono? Plus, MS has no interest in seeing .NET succeed on any platform other than Windows. In fact, they can't have it happen.

The weird thing is that from what I can see, more people are using VB.NET than C#.

Mark
via e-mail

IDEs Generally Lead to Better Productivity



I wonder if all those so-called hard-core programmers (who proudly proclaim that Notepad is better than any IDE) ever bothered to stop and think about what they were justifying ("Is the Graphical IDE a Good Thing" [Vol. 7, issue 3]).

Basically, what they're saying is that it's better to continually rewrite the basic plumbing than to use "intellisense" or some other feature that an IDE provides, which takes care of the banal and lets you focus on the task at hand. I would much rather focus on the business rules I have to implement than edit a deployment descriptor by hand, or write the basic Java code for the GUI of a Swing application. What I care for most is that the meat of my app performs well and, most important, adheres to relevant business rules.

Don't get me wrong, I learned Java with TextPad, but I became much more productive in NetBeans than I ever was in TextPad.

I wonder what those "hard-core" code-gurus will say when they see how quick it is to publish a Web service with Visual Studio .NET versus the hours of coding it would take to do it by hand in Java. I'm sure Java IDEs will do the same in time.

Reuben Cleetus
rcleetus@yahoo.com

IDEs Have Their Own Niche

Coming from my personal experience I would prefer a powerful text editor to any IDE, especially in the Java world. The problem with an IDE is you have to learn it, and it has bugs and perks of its own to overcome. When you're a beginner, an IDE is perfect with all its wizards and graphical buttons, etc. As you become more professional, you stop using wizards and feel uncomfortable with limited text-editing functions and doing the miscellaneous things the IDE doesn't do. Once you're a pro, you throw them away and take Emacs/Ant and live happily ever after.

Admiral
admiralspb@yahoo.com

First of all, IDE stands for "Integrated Development Environment," not "Interactive." So what if there are wizards? Most experienced engineers never use them anyway. The real benefit to an IDE is the productivity – all your editing, debugging, compiling, etc., are "integrated" (hence "IDE") into a single environment. This is much more productive than crawling on your hands and knees with a text editor and command-line debuggers and the like, jumping around in various utilities, etc.

Scott
splunge_000@yahoo.com

Java Anthem

In the April issue of *JDJ*, I made a call to readers to come up with a possible Java song, something that we could all maybe hum and sing along to. We received some really great responses, but this one seemed the best.
–Alan Williamson

The melody is "Nobody Does It Better" by Marvin Hamlisch, from the James Bond movie The Spy Who Loved Me.

"The Language That Loved Me"

No code does it better
Makes me feel sad for the rest.
No code does it half as good as you.
Java, you're the best.

I wasn't looking but somehow you found me.
I tried to hide from your interpreter,
Like heaven above me, the language that loved me
Is keeping all my software safe tonight.

And no code does it better
Though sometimes I wish C# could.
No code does it quite the way you do.
Did you have to be so good?

The way that you recover whenever you crash.
There's some kind of magic inside you
That keeps me from VB but just keep it easy.
How did you learn to debug the way you do?

And no code does it better
Makes me feel sad for the rest.
No code does it half as good as you.
Java, Java you're the best.
Java you're the best.
Java you're the best.
Java you're the best.
Java you're the best.
Java you're the best.
Java you're the best.
Java, Java you're the best.

submitted by John Chamberlain
jcpublic@attbi.com

Missed an issue? *We've got 'em all for you on CD!*

JDJ The Complete Works	Web Services Resource CD	CFDJ The Complete Works	CFA The Complete Works
~~Reg $119.99~~	~~Reg $119~~	~~Reg $79.99~~	~~Reg $79.99~~
Buy Online Only $71.99	Buy Online Only $79	Buy Online Only $71.99	Buy Online Only $71.99

SPECIAL OFFER: Buy CFDJ & CFA The Complete Works For Only $129.99

COLLECT ALL 4 FOR ONLY $229.99 JDJ, WSJ & XML-J, CFDJ, & CFA

JDJStore.com

Order Online and Save 10% or More! WWW.JDJSTORE.com

OFFER SUBJECT TO CHANGE WITHOUT NOTICE

SYS-CON Media, the world's leading publisher of *i*-technology magazines for developers, software architects, and e-commerce professionals, brings you the most comprehensive coverage of WebSphere.

SYS-CON MEDIA

WebSphereDevelopersJournal.com

WebSphere DEVELOPER'S JOURNAL

Introductory Charter Subscription

SUBSCRIBE NOW AND SAVE $31.00 OFF THE ANNUAL NEWSSTAND RATE

ONLY $149 FOR 1 YEAR (12 ISSUES) REGULAR RATE $180

OFFER SUBJECT TO CHANGE WITHOUT NOTICE

Do You Have Access to the Internet?

The World's Leading Independent WebSphere Developer Resource

Then Subscribe Online and Save $31! It's that easy

SYS-CON's Web Services Edge World Tour Attendance Quadruples in San Francisco

(Montvale, NJ) – SYS-CON Events, Inc., presented the fourth stop of their Web Services Edge World Tour, SYS-CON's popular one-day tutorial series, to a sold-out audience in San Francisco. The series was organized in response to developers' eagerness to plan ahead for SYS-CON's Web Services Edge 2002 International Conference & Expos East and West, which will take place June 24–27 in New York City and October 1–3 in San Jose, CA, respectively (www.sys-con.com/WebServicesEdge2002East).

The first four cities of the Web Services Edge World Tour tutorial series entitled "Developing SOAP Web Services" are exclusively sponsored by Systinet Corporation (www.sys-con.com/education). The tutorial curriculum is designed to equip professional developers with all the tools and information they need to immediately begin creating, deploying, and using their own Web services. Upon completion of the program, expert practitioners will be able to take an applied approach and cover base technologies such as SOAP, WSDL, UDDI, and XML, as well as more advanced topics such as security, J2EE integration, exposing legacy systems, and integrating Web services into an existing IT infrastructure.

Due to an overwhelming interest in the tutorial in San Francisco, SYS-CON Events increased the originally planned single-session program to four simultaneous sessions of which two were presented by Oracle and focused on the topic of "Architecting J2EE Web Services."

SYS-CON Offers New Opportunities in San Francisco

SYS-CON Events also brought leading Web services vendors and the attendees of the Web Services Edge World Tour together. More than 800 attendees of San Francisco's one-day tutorial were excited to meet with IONA, AltoWeb, Sonic Software, ObjectFocus, Systinet, and Oracle representatives. They received a unique chance to ask questions and examine these companies' Web services offerings.

Next Stop: Thursday, June 27, 2002, New York City

The next stop of the education tour is June 27, 2002, in New York City. There will be 12 new cities named before the end of this year, covering additional stops in North America and introducing several new cities in Europe, Asia, and Australia.

www.sys-con.com

FULCRUM NOW MAKES THE U.S. SCENE

(Pune, India) – Software tools company AccelTree has debuted FULCRUM, an intelligent Java code builder, in the United States. Neither an IDE nor a code generator, FULCRUM is a Java development tool that uses a proprietary concept of code templates that can be used as building blocks to construct Java objects and applications.

Among the reported benefits are faster coding time and the ability to implement your own coding standards. In addition, the need for skilled programmers is reduced, and no runtime software needs to be purchased. The tool also integrates with existing or external objects.

www.AccelTree.com

REMOTEAPPS LAUNCHES XYRIAN 2.2

(London) – RemoteApps has announced the launch of Xyrian 2.2 Enterprise and Developer Editions, supported by an advanced Tech Portal.

Xyrian, an application development tool, features an advanced programming environment, the Framework persistence engine, and Team View, a browser-based content management system.

www.remoteapps.com

Quintessence and Cape Clear Announce Alliance

(Berkshire, UK / Dublin, Ireland) – Quintessence Systems and Cape Clear Software will work together to enable customers to migrate and connect existing Oracle applications to Web services both inside the corporate network and across the Internet. Customers can use Quintessence's in2j tool to automatically migrate their Oracle legacy code into Java. Once in Java, Cape Clear's CapeConnect platform can be used to expose the migrated Java code as Web services.

www.in2j.com
www.capeclear.com

SYS-CON Media

www.sys-con.com

JAVA DEVELOPER'S JOURNAL

The World's Leading Java Resource

HOME | ADVERTISE | SUBSCRIBE | CONTACT | SYS-CON MEDIA www.javadevelopersjournal.com

J2ME | J2SE | J2EE | Home | Online

DOT.COM
- buyer's guide
- java forums
- mailing list
- java jobs
- java store

MAGAZINE
- advertise
- authors
- customer service
- editorial board
- subscribe

CONTENT
- archives
- digital edition
- editorial
- features
- interviews
- product reviews
- source code

upcoming events

web services EDGE conference & expo EAST

JDJ EDGE conference & expo

XML EDGE conference & expo 2002

JUNE 24–27 2002
NEW YORK
NEW YORK

web services EDGE conference & expo WEST

XML EDGE conference & expo 2002

JDJ EDGE conference & expo

wireless EDGE conference & expo

OCT 1–3 2002
SAN JOSE
CALIFORNIA

What's Online...

June 2002

JavaDevelopersJournal.com

Visit www.javadevelopersjournal.com and learn about the latest news and events from the world's leading Java resource. Know what's happening in the industry, minute by minute, and stay ahead of the competition.

Subscribe to Our Free Weekly Newsletters

Now you can have the latest industry news delivered to you every week. SYS-CON newsletters are the easiest way to keep ahead of the pack. Register for your *free* newsletter today! There's one for Java, XML, Web services, Wireless, and ColdFusion. Choose one or choose them all.

Search Java Jobs

Java Developer's Journal is proud to offer an employment portal for IT professionals. Get direct access to the best companies in the nation and discover the "hidden job market." If you're an IT professional curious about the job market, this is the site to visit.

Simply type in the keyword, job title, and location and get instant results. You can search by salary, company, or industry.

Need more help? Our experts can assist you with retirement planning, putting together a résumé, immigration issues, and more.

Web Services Edge 2002 Conference and Expo

Sign up for the Web Services Edge 2002 West International Conference and Expo, October 1–3, at the San Jose Convention Center, San Jose, California. This year's event is expected to be the biggest and best ever with the top IT professionals in the business speaking about the hottest industry topics. The Web Services Edge 2002 Conference and Expo will feature presentations prepared for a diverse audience. Whether you consider yourself a beginner or at the top of the IT world, this conference is for you!

2002 Readers' Choice Awards

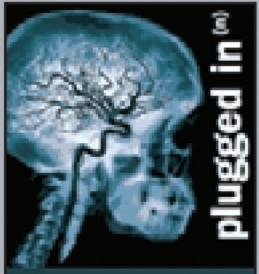
Make your opinion count. Vote today for the *Java Developer's Journal* Readers' Choice Awards, which have earned the reputation as the most respected industry awards of its kind. Known as the "Oscars of the Software Industry," this year's awards feature an expanded list of product categories and other additions that will make the fifth annual Readers' Choice Awards the best so far!

JavaBuyersGuide.com

JavaBuyersGuide.com is your best source on the Web for Java-related software and products in more than 20 mission-critical categories, including application servers, books, code, IDEs, modeling tools, and profilers. Check the Buyer's Guide for the latest and best Java products available today.

Full version of the new Oracle9*i* JDeveloper

Full version FREE



THE JAVA DEVELOPER CONNECTION℠ PROGRAM

Get pre-release Java™ technology downloads

REGISTER TO GET THEM

JAVA Sun

WAKESOFT Learn more now!

Click for a FREE 30-day trial AltoWeb

macromedia JRUN

sitraka

Get SonicMQ® sonic SOFTWARE

MERANT DataDirect

Get a Free Java™ Developers Solution CD. Rational®XDE™ Rational

CLICK HERE NOW SilverStream eXtend Web Services

spiritsoft going beyond jms

InstallAnywhere PowerUpdate ZERO G www.ZeroG.com Click here to try

JDJ Store

www.jdjstore.com

Contract-to-Hire

That was then, this is now

WRITTEN BY
Bill Baloglu &
Billy Palmieri

As staffing professionals in the technology industry, we've seen the focus for many staffing firms shift from placing senior consultants in contract positions (while occasionally filling the odd full-time job) to placing them in full-time jobs.

Over the past several years, companies felt that paying contract consultants a high hourly rate to execute one piece of a project was a great way to get the job done without committing to the salary, benefits, and responsibilities that go along with hiring a full-time employee.

But that, as they say, was then. And this is now.

This may change again as the economy and the industry readjust in the coming months, but for the time being, companies are looking to fill full-time positions. Period.

Many of you who are reading this are seasoned contract consultants. You've invested time and money into becoming incorporated, and you've grown accustomed to the benefits of being a "hired gun" contractor.

Working on two or three contracts a year has exposed you to many more types of technologies than you would have worked with as a full-timer at just one company. And while full-time employees have grappled with corporate dynamics and politics, you've been able to walk away from it all when your contract was finished.

However, if you're looking to make the change from a contractor to a full-time employee, there are several issues you need to overcome in order to be considered a viable full-time candidate.

When managers look at a long-time contractor's résumé, they raise some of the following objections, and we've provided some points you can respond with.

1. ***I don't want to hire a contractor because when the contract market opens back up, he or she will be the first one gone.***

Managers are under the illusion that they'll be able to keep people in full-time positions, but full-time employees are just as likely to leave a job if the work is not challenging. The average full-time employee in Silicon Valley lasts 18 months in one job.

Convince the manager that if the work is challenging and interesting, you'll stay. But if it's not, then the average full-time employee isn't likely to stay either.

Your résumé is important. A lot of three-, four-, or five-month contracts on a résumé can lead managers to wonder if the contracts were actually completed, or why they weren't renewed.

Nine-month to one-year contracts look good. They suggest that your work was valued and that you stayed on as long as was necessary. It's always good to be able to tell managers that you never left a contract before it was finished.

2. ***Contractors are used to coming in and working for a limited time on just one piece of the project. They don't have a hands-on understanding of the full life cycle of a project.***

On the issue of project life cycles, it could be helpful to bring up the following points.

As a contractor you may not have been around when the project was architected and developed or when it was time to fix bugs and release it, as most people don't hire a contractor until they have a problem.

The experienced contractor is often required to review the project's beginning stages, from its requirements gathering to issues of scalability and security, to find the problem he or she was hired to solve. In that respect you may very well have a strong understanding of project life cycles.

3. ***Contractors don't get involved in a company's internal process and don't usually deal with issues of internal politics. But I need someone with that experience.***

You may not have dealt with the internal politics of a company, but a savvy contractor knows that he or she will be dealing and communicating with different people on the project in order to examine the problems.

Many contractors are typically brought in once the project is in trouble and internal tensions are high, so the internal staff is eager to avoid blame for the problem. You could make a case for your experience in handling internal politics on that level.

4. ***Contractors are used to making a lot more money than I can pay.***

In a full-time position, you won't be earning $125 an hour (as you may have as a contract consultant). You're more likely to earn $125K a year plus benefits. A senior position could pay more, but don't count on it. Make it abundantly clear to the hiring manager that you're aware of these economic realities and you're okay with them.

If you're not, feel free to keep holding out until it changes back to a contractor's world.

jdjcolumn@objectfocus.com

AUTHOR BIOS
Bill Baloglu is a principal at ObjectFocus (www. ObjectFocus .com), a Java staffing firm in Silicon Valley. Bill has extensive OO experience and has held software development and senior technical management positions at several Silicon Valley firms.

Billy Palmieri is a seasoned staffing industry executive and a principal at ObjectFocus. His prior position was at Renaissance Worldwide, where he held several senior management positions in the firm's Silicon Valley operations.

JAVA DEVELOPER'S JOURNAL ADVERTISER INDEX

ADVERTISER	URL	PHONE	PAGE
AccelTree	www.acceltree.com		21
Actuate Corporation	www.actuate.com/info/jbjad.asp	800-884-8665	57
Altaworks	www.altaworks.com	603-598-2582	73
Altova	www.altova.com		39
AltoWeb	www.altoweb.com		35
Apple Computer, Inc.	www.apple.com/macosx	1-800-MY-APPLE	4-5
Ashnasoft Corporation	www.ashnasoft.com		45
BEA	www.bea.com/download		6
Borland Software Corp.	www.borland.com/new/jb7/5068.html	800-252-5547	63
Canoo Engineering AG	www.canoo.com/ulc/	41 61 228 94 44	31
Capella University	www.capellauniversity.edu	1-888-CAPELLA	40
Compuware Corp.	www.compuware.com/products/optimalj	800-468-6342	25
CTIA WIRELESS I.T.	www.ctiashow.com		83
Dice	www.dice.com		49
EnginData Research	www.engindata.com	201-802-3082	93
Fiorano Software	www.fiorano.com/tifosi/freedownload.htm	800-663-3621	71
HiT Software	www.hitsw.com/xmlrdb	408-345-4001	59
InetSoft Technology Corp.	www.inetsoft.com/jdj	888-216-2353	65
Infragistics, Inc.	www.infragistics.com	800-231-8588	14-15
InstallShield Software Corp.	www.installshield.com	847-240-9111	55
INT, Inc	www.int.com	713-975-7434	26
iSavvix	www.community.isavvix.com	866-ISAVVIX	85
JDJEdge Conference & Expo	www.sys-con.com		95
JDJ Store	www.jdjstore.com	201-802-3012	107,111
Jinfonet	www.jinfonet.com		79
LOOX Software Inc.	www.loox.com	800-684-LOOX	53
Macromedia	www.macromedia.com/go/jrun4jdj		23
Metrowerks Corp.	www.wireless-studio.com		11
\n software inc.	www.nsoftware.com		29
New Atlanta Communications	www.newatlanta.com		43
Northwoods Software	www.nwoods.com/go/	800-434-9820	48
Parasoft Corporation	www.parasoft.com/jdj6	888-305-0041	47
Pramati Technologies	www.pramati.com	877-PRAMATI	75
QUALCOMM Incorporated	http://brew.qualcomm.com/ZJD5		67
Quintessence Systems Limited	www.in2j.com		41, 81
Rational Software	www.rational.com/offer/javacd2		37
Rational User Conference 2002	www.rational.com/ruc	888-889-0074	77
RemoteApps	http://portal.remoteapps.com	44(0)20 7680 7689	17
SilverStream Software	www.silverstream.com/scale	1-888-823-9700	27
Sitraka	www.sitraka.com/jclass/jdj	800-663-4723	19
Sitraka	www.sitraka.com/jprobe/jdj	800-663-4723	61
Sitraka	www.sitraka.com/performasure/jdj	800-663-4723	116
Simplex Knowledge Company	skc.com	845-620-3700	89, 91
Sonic Software	www.sonicsoftware.com	800-989-3773	2
SpiritSoft	www.spiritsoft.com/climber		33
Sprint PCS	http://developer.sprintpcs.com		69
Sun Microsystems	www.sun.com/forte		13
SYS-CON Industry Newsletters	www.sys-con.com	201-802-3020	87
SYS-CON Subscription Offer	www.sys-con.com/suboffer.cfm		109
TogetherSoft Corporation	www.togethersoft.com/challenge/1	919-833-5550	8
WebGain, Inc.	www.webgain.com/toplink_create3.html	1-877-WebGain Ext.15858	115
WebLogic Developer's Journal	www.sys-con.com/weblogic	800-513-7111	109
Web Services Edge Conference & Expo	www.sys-con.com		104-105
Web Services Edge World Tour 2002	www.sys-con.com	201-802-3069	108
Web Services Resource CD	www.jdjstore.com	201-802-3012	99
Web Services Journal	www.sys-con.com	800-513-7111	100
WebSphere Developer's Journal	webspheredevelopersjournal.com	800-513-7111	107
Zero G	www.zerog.com	415-512-7771	3

General Conditions: The Publisher reserves the right to refuse any advertising not meeting the standards that are set to protect the high editorial quality of ***Java Developer's Journal***. All advertising is subject to approval by the Publisher. The Publisher assumes no liability for any costs or damages incurred if for any reason the Publisher fails to publish an advertisement. In no event shall the Publisher be liable for any costs or damages in excess of the cost of the advertisement as a result of a mistake in the advertisement or for any other reason. The Advertiser is fully responsible for all financial liability and terms of the contract executed by the agents or agencies who are acting on behalf of the Advertiser. Conditions set in this document (except the rates) are subject to change by the Publisher without notice. No conditions other than those set forth in this "General Conditions Document" shall be binding upon the Publisher. Advertisers (and their agencies) are fully responsible for the content of their advertisements printed in ***Java Developer's Journal***. Advertisements are to be printed at the discretion of the Publisher. This discretion includes the positioning of the advertisement, except for "preferred positions" described in the rate table. Cancellations and changes to advertisements must be made in writing before the closing date. "Publisher" in this "General Conditions Document" refers to SYS-CON Publications, Inc.

Next Month

Programming Restrictions in EJB Development

Programming restrictions can have a significant impact on your usual coding techniques. Leander van Rooijen reviews some of the restrictions as defined in the EJB specifications 2.0 and offers an alternative approach.

Combating the 'Object Crisis'

The phenomenal success of Java as an enabler of enterprise-wide, Web-deployed applications has compelled a countless number of organizations and individuals alike to seek proficiency with Java. Jacquie Barker notes that many, however, are ill-prepared to harness Java's power as an object-oriented programming language due to a lack of understanding of object concepts.

Certificate Authorization in Your J2EE PKI

Eric Simmerman demonstrates how he built a PKI implementation in which his client acted as the Certificate Authority. He then integrated this PKI with J2EE systems via JSSE to provide secure communication services among multiple platform boxes distributed across three continents.

Optimizing Java Performance in Heritage Designs

Dr. Carl Barratt discusses the introduction of Java into multilanguage heritage designs, focusing on the advantages and disadvantages of deploying each solution.

'Subversion, One Point Oh'

WRITTEN BY
BLAIR WYMAN

Sometimes I think the world is getting fundamentally goofier, at an ever-increasing pace. On the other hand, I've only been here a brief while (in geologic terms, at least), and can't help believing that the world has always been pretty doggone goofy.

Sputnik and I happened to arrive the same year, 1957, though Sputnik made a much bigger splash in the press. Here is a testament to the magnitude of orbital velocity: a 183-pound device, accelerated into Earth orbit, can drag an entire surprised world right along with it. F=ma, indeed.

At the time a key parameter in the acceleration of global goofiness was surely the simultaneous acceleration of the space race. We learned to count backwards – "T-minus" one second at a time – most of us blissfully unaware of the meaning of "T". One of the first things I vividly remember is watching an Atlas-6 rocket lift John Glenn into Earth orbit. My equilibrium had been forever punctuated. (I'd never seen a color TV before.)

There was a lot happening during those goofy years, and not all the voices were counting down in synchronized unity. While it's true that conformity was king, the muzzles of McCarthyism were loosening. Diverse voices once again posed probing questions, flatly eschewing pat answers. Of course, as soon as someone uses the word "eschew," half their audience is history. How, then, to make people listen?

It seems that if you make people laugh, they will listen to you. After all, laughing is fun; it feels good. So, if you happen to have a particularly outrageous – or even subversive – point to make, why not make it with humor?

It was a Saturday afternoon when buoyant humor rescued me from what had, so far, been a pretty mainstream float through childhood. Flipping through the channels (I think we got five or six), I happened upon a starkly rendered warning:

It is the stated position of the U.S. Air Force that their safeguards would prevent the occurrence of such events as are depicted in this film…

And so began Stanley Kubrick's dark comedy, *Dr. Strangelove, or: How I Learned to Stop Worrying and Love the Bomb.*

On the surface, it's an unsettlingly scary premise: a plausible scenario, in which human frailty leads to nuclear Armageddon. (I am reminded of that "Buffy" episode, in which Giles says, "It's the end of the world. Everyone dies. It's rather important, really.") So why was it so outrageously funny? One possible explanation springs to mind: "goofiness."

Some years later, during that brief and shining time when I knew it all, my taste in music was singularly insular. Music, of all things, certainly wasn't funny. Rather, music was a profoundly ponderous accompaniment to the equally grave business of unbridled teenage lust: an ear-pounding backdrop to the perpetual pursuit of pulchritude.

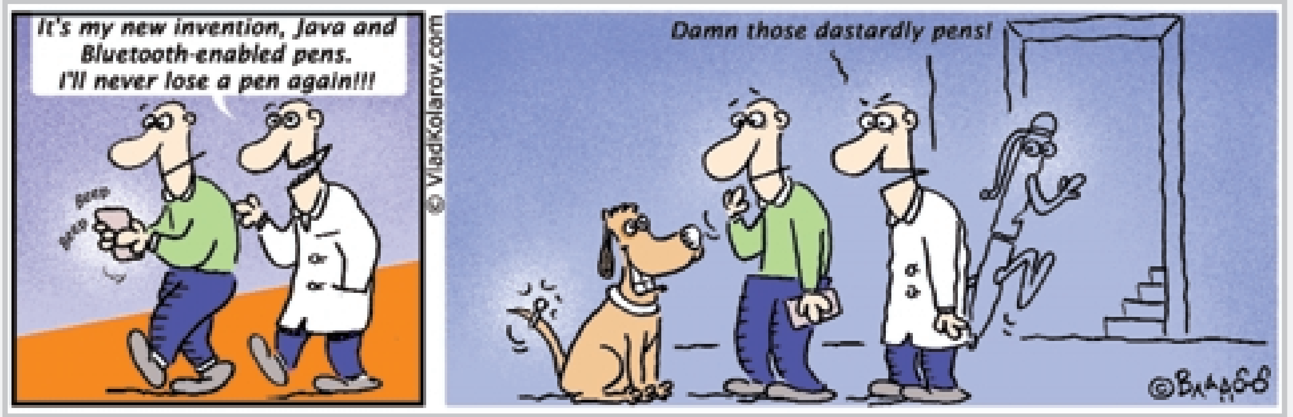
Then, a great friend of the family introduced me to the musical genius of Tom Lehrer. I had found the contrapuntal companion to Strangelove, poking furtive fun at soberly spooky specters like irreverent Boy Scouts ("Be Prepared"), Dahmer-esque dismemberment ("I Hold Your Hand in Mine"), and nuclear conflagration ("We'll All Go Together When We Go"). That familiar melody, with a new lyric – "Down by the old maelstrom…" – inevitably evokes an irrepressible grin.

So, dear reader, I implore you: don't fear goofiness, embrace it. After all, it has been with you your whole life and always will be, so laugh at it when you can. In the words of Mary H. Waldrip, "A laugh is a smile that bursts."

blair@blairwyman.com

AUTHOR BIO
Blair Wyman is a software engineer working for IBM in Rochester, Minnesota, home of the IBM iSeries.

Java Dudes


It's my new invention, Java and Bluetooth-enabled pens. I'll never lose a pen again!!!
beep
beep
© Vladkolarov.com
Damn those dastardly pens!
©BLAGOS

WebGain,Inc.

www.webgain.com/toplink_create3.html

Sitraka

www.sitraka.com/performasure/jdj