// JANUARY/FEBRUARY 2013 /

Java™ magazine

By and for the Java community

EMBEDDED EVERYWHERE

THE FUTURE OF CONNECTED DEVICES STARTS WITH JAVA

ORACLE.COM/JAVAMAGAZINE

ORACLE

//table of contents /

COVER ART BY I-HUA CHEN, NICHOLAS PAVKOVIC

The age of embedded computing is here. Analysts predict that within this decade, we'll see tens of billions of embedded computational devices entering our daily lives. And the possibilities for these "smart" devices are literally endless: connected vehicles, appliances, utility meters, medical devices, industrial controllers, and even a contraption that monitors how much alcohol your local bartender pours into your cocktail.

In this embedded-focused issue, we explore why Java is the best language choice for embedded development. In "Embedded Everywhere," Oracle's Terrence Barr talks about why Java's write once, run anywhere technology is perfectly positioned for the coming wave of embedded devices. He sees Java in the embedded space creating an open, standardized technology infrastructure. What's more, Java offers 9 million developers who can apply their knowledge to embedded development. Barr talks about getting started with embedded development using a Keil board and also explores what embedded device growth will mean for big data.

In "Top Ten Reasons for Using Java in Embedded Apps," Simon Ritter makes a case for Java as the go-to language for embedded. "We already have a wheel," he says. "Let's not keep inventing new ones."

We also bring you two stories of embedded Java in action. In "Java Arrives on a $25 Board," we talk with Raspberry Pi Foundation Cofounder Eben Upton about the low-cost programmable computer and Java's role. In "The Future of Money," we explore the emerging world of digital currency and introduce you to the Royal Canadian Mint's digital currency, MintChip, which runs on the Java Card platform.

The embedded space presents exciting opportunities for Java developers. It's a new year, and a really great time to be a Java developer.

Caroline Kvitka, Editor in Chief BIO

//send us your feedback /

We'll review all suggestions for future improvements. Depending on volume, some messages may not get a direct reply.

PHOTOGRAPH BY BOB ADLER

EDITORIAL

Editor in Chief
Caroline Kvitka

Community Editors
Cassandra Clark, Sonya Barry, Yolande Poirier

Java in Action Editor
Michelle Kovac

Technology Editors
Janice Heiss, Tori Wieldt

Contributing Writer
Kevin Farnham

Contributing Editors
Claire Breen, Blair Campbell, Karen Perkins

DESIGN

Senior Creative Director
Francisco G Delgadillo

Senior Design Director
Suemi Lam

Design Director
Richard Merchán

Contributing Designers
Jaime Ferrand, Nicholas Pavkovic

Production Designers
Sheila Brennan, Kathy Cygnarowicz

PUBLISHING

Vice President
Jeff Spicer

Publisher
Jennifer Hamilton +1.650.506.3794

Audience Development and Operations Director
Karin Kinnear +1.650.506.1985

ADVERTISING SALES

Associate Publisher
Kyle Walkenhorst +1.323.340.8585

Northwest and Central U.S.
Tom Cometa +1.510.339.2403

Southwest U.S. and LAD
Shaun Mehr +1.949.923.1660

Northeast U.S. and EMEA/APAC
Mark Makinney +1.805.709.4745

Advertising Sales Assistant
Cindy Elhaj +1.626.396.9400 x 201

Mailing-List Rentals
Contact your sales representative.

RESOURCES

Oracle Products
+1.800.367.8674 (U.S./Canada)

Oracle Services
+1.888.283.0591 (U.S.)

Oracle Press Books
oraclepressbooks.com

ARTICLE SUBMISSION

If you are interested in submitting an article, please e-mail the editors.

SUBSCRIPTION INFORMATION

Subscriptions are complimentary for qualified individuals who complete the subscription form.

MAGAZINE CUSTOMER SERVICE

java@halldata.com **Phone** +1.847.763.9635

PRIVACY

Oracle Publishing allows sharing of its mailing list with selected third parties. If you prefer that your mailing address or e-mail address not be included in this program, contact Customer Service.

Java Magazine is published bimonthly with a free subscription price by Oracle, 500 Oracle Parkway, MS OPL-3C, Redwood City, CA 94065-1600.

Digital Publishing by Texterity


Henrik Stahl

The Java band entertains the crowd at the Community keynote.


Georges Saab

JAVAONE LOOKS FORWARD

JavaOne Brazil took place in São Paulo, Brazil, December 4–6. Latin America is an important development hub, and it looks as though JavaOne Brazil, now in its third year, is here to stay. "We continually come back to Latin America because of the dedication the community has to driving continued innovation for Java," said Oracle's **Henrik Stahl** in kicking off the Java Strategy and Java Technical keynotes on the first day of the conference. He noted that the success of Java depends on technological innovation, strong stewardship from Oracle, and community participation. "The Latin American Java community (especially in Brazil) is a shining example of how to be a positive contributor to Java," he said.

Oracle's **Georges Saab** discussed some of the recent and upcoming changes to Java. "In addition to the incremental improvements to Java 7, we have also increased the set of platforms supported by Oracle from Linux, Windows, and Oracle Solaris to now also include Mac OS X and Linux/ARM for ARM-based PCs such as the Raspberry Pi and emerging ARM-based microservers."

Oracle's **Staffan Friberg** provided an overview of some changes coming in Java 8, including lambda expressions, removal of the permanent generation (PermGen) heap, improved date and time APIs, and improved security.

Oracle's **Judson Althoff** started his talk off with a bang. "The Internet of Things is on a collision course with big data, and this is a huge opportunity for developers," he said. Althoff noted that a car embedded with sensors for fuel efficiency, temperature, and tire pressure can generate a petabyte of data a day.

When a "blue screen of death" appeared for the Technical keynote, presenters gladly went on without their slides. What followed was a collection of demos, including JavaFX on a tablet and a look at Project Easel in NetBeans.

Throughout the conference, attendees chose from dozens of sessions, more than half of which were selected by the community, and six hands-on labs.

PHOTOGRAPHS BY LUCIANA AITH, ORQUESTRA DE IMAGENS

DUKE'S CHOICE LAD WINNERS HONORED

Top: TQTVD's David Britto with Stephen Chin (left), Bruno Souza, and Yara Senger. Center: TIVIT's Einar Saukas and Fabiano Cury with Souza, Chin, and Senger. Bottom: CPqD Foundation's Sonia Mayumi Kutiishi and Hugo Cesar with Souza, Chin, and Senger.

PHOTOGRAPHS BY LUCIANA AITH, ORQUESTRA DE IMAGENS

At JavaOne Brazil, winners of the first regional Duke's Choice Awards were recognized for their innovative use of Java.

The winners of the Duke's Choice Awards LAD are TQTVD Software, TIVIT Software, and CPqD Foundation. While all of the winners hail from Brazil, nominations were received from across the region. Nominees included NeoTropic's Kuwaiba project and Vortexbird's Zathura Code Generator project happening in Colombia. Judges included **Yara** and **Vinicius Senger** of SouJava Brazil and **Alexis Lopez** of ColombiaJUG.

HERE'S A LOOK AT EACH WINNER:

TQTVD Software's AstroTV project is the Brazilian middleware compliant with the Ginga specification that allows full interactivity, extended content programming, and portability. The Java-based Ginga specification is the intermediate software layer that enables the development of interactive applications for digital TV independently from the hardware platform of digital receivers manufacturers (set-top boxes). AstroTV has been embedded in more than 3 million devices.

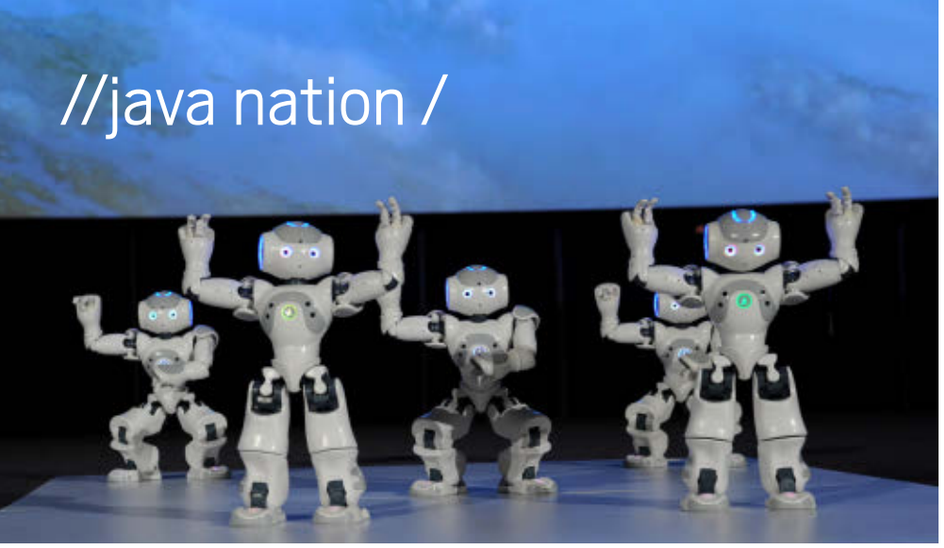
TIVIT Software's Central de Cessão de Crédito (C3) project is a Java-based high-performance scalable transactional distributed system that intermediates and manages all credit assignment operations between financial institutions in Brazil and tracks credit ownership. It was developed as a result of BACEN (the Central Bank of Brazil) identifying irregular credit assignment operations between financial institutions that created a deficit of R$4.3 billion in the Brazilian banking system.

CPqD Foundation's SMTVI (Multiplatform Services for Digital Interactive TV) project dramatically improved the digital literacy of the majority of the Brazilian population. This population has insufficient financial resources to keep up with new technological developments; TV is its only technology resource. The project included the creation of interactive digital TV applications in the areas of health, retirement, employment, chat, weather, news, e-commerce, education, and games. The use of Java technology allowed the design of a distributed, scalable, and service-based architecture.

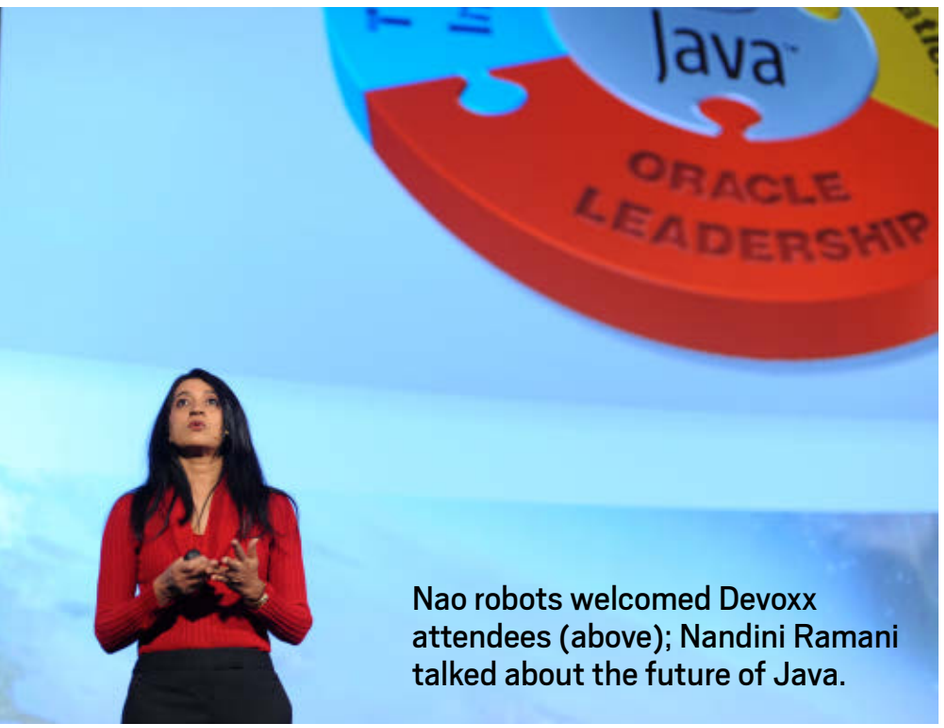
Geeks Ride Again

Java enthusiasts donned bike helmets and Duke's bike jerseys on the Saturday before JavaOne Brazil for a riding tour of São Paulo. At this now-traditional preconference ride, more than 20 participants enjoyed cycling from Bicycle Park through downtown on street lanes closed off for bicyclists. The ride was 30 kilometers, but participants could opt to take the subway for part of the trip.

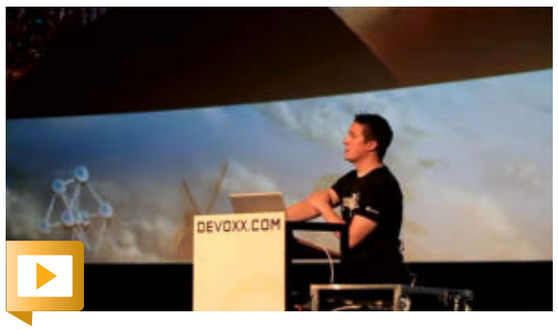
PHOTOGRAPH BY TORI WIELDT



DEVOXX 2012 INNOVATES



Nao robots welcomed Devoxx attendees (above); Nandini Ramani talked about the future of Java.



Catch the sights and sounds of Devoxx 2012.

PHOTOGRAPHS COURTESY OF DEVOXX

Devoxx, the biggest Java conference in Europe, hosted 3,400 attendees from 40 countries November 14–16 in Brussels, Belgium. Through word of mouth, it sold out six weeks in advance and had hundreds of developers on the waiting list. The conference is "for developers, by developers," and that's the key to its success, according to attendees. In 200 technical sessions, 190 well-known Java community experts talked about the latest products, frameworks, Java language improvements, and more.

Conference innovation included a room display monitor infrastructure that used Oracle Java SE Embedded running on a Raspberry Pi and a rating application for attendees to like or dislike sessions. Oracle's **Jasper Potts** created an interactive schedule using JavaFX for attendees' convenience. Parleys, the Java e-learning Website, was redesigned with HTML5 and GlassFish, and allowed speakers to post their slides online before their talks and then sync the soundtracks with the slides. During the conference, attendees participated in hands-on workshops and enjoyed coding with their peers in the hackergarten, which hosted **Stephen Chin**'s NightHacking and a live coding project called Code Story.

The keynote was introduced by five dancing humanoid Nao robots, setting the stage for a talk by Oracle's **Nandini Ramani** about technical advances and the latest Java embedded technologies. "Java continues to drive the applications and devices that enrich our interactivity with the world around us," she said.

Oracle technologists presented details of improvements to the Java platform. **Joe Darcy** and **Dalibor Topic** discussed the latest update to JDK 7 and the upcoming features in JDK 8. **Jasper Potts**, **Jonathan Giles**, **Jim Weaver**, and **Michael Heinricks** presented eight talks on effective client development with JavaFX using its UI controls, TableView, and Scene Builder. **Simon Ritter**, **Stephen Chin**, and **Daniel Blaukopf** talked about programming and front-end development on devices with Java embedded. **Shaun Smith**, **Marek Potociar**, **Arun Gupta**, **Jitendra Kotamraju**, and **David Delabasse** presented the evolution of Java persistence, JSON processing, JAX-RS, and WebSocket in a series of sessions over the three days.

DEVOXX4KIDS

The first Devoxx4Kids one-day coding workshops took place October 13 and October 20 in Brussels, Belgium (in Flemish and French). One of the speakers was a Nao robot that talked about the fun of learning programming and showed off its karate moves. Sixty students age 11 to 14 attended three sessions where they programmed Lego Mindstorm and Mars Rover robots and coded with Scratch, a drag-and-drop programming interface to create interactive stories and games.

Stephan Janssen, the organizer behind the popular Devoxx Java conference, was inspired to organize this event by his son. "My 11-year-old son wanted to learn programming, and I could not find any content in Flemish," he said. He partnered with **Tasha Carl**, who organized robotic workshops with Greenlight for Girls, an organization encouraging young girls to consider a future in science, technology, engineering, or math.

At the Devoxx keynote in November, Janssen announced that Java user groups and developers can use his content and Web infrastructure to replicate Devoxx4Kids in their regions. "The workshops were really fun and such a rewarding experience," he said.

So far, Devoxx4Kids events are being organized in the Netherlands, the UK, and France.

PHOTOGRAPHS COURTESY OF DEVOXX; TRIDIUM BY ORANGE PHOTOGRAPHY

Tridium Wins Oracle Excellence Award

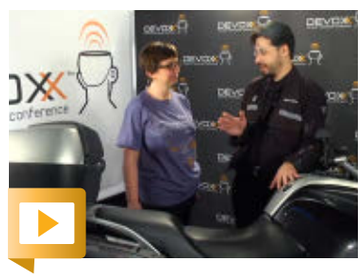
Tridium's John Sublett (left) with Oracle's Judson Althoff

Tridium, which develops solutions for connecting devices to the enterprise, has won the 2012 Oracle Excellence Award for Java Business Innovation. The company's Java-based Niagara Framework integrates diverse systems and devices, regardless of communication protocol, into a unified platform that can be easily managed and controlled in real time over the internet using a standard Web browser. The framework facilitates remote energy metering, home automation, and industrial process management.

Tridium was recognized for using Java embedded technology and Java SE to create new possibilities for addressing how devices and systems connect and interact with each other, as well as with the core enterprise platform. Its Niagara Framework has been implemented in varied application environments across many different business sectors. Tridium has applied Java's entire platform spectrum, from embedded to enterprise, to create a uniform software environment that integrates microdevices with each other and links them back to the enterprise applications of which they are components.

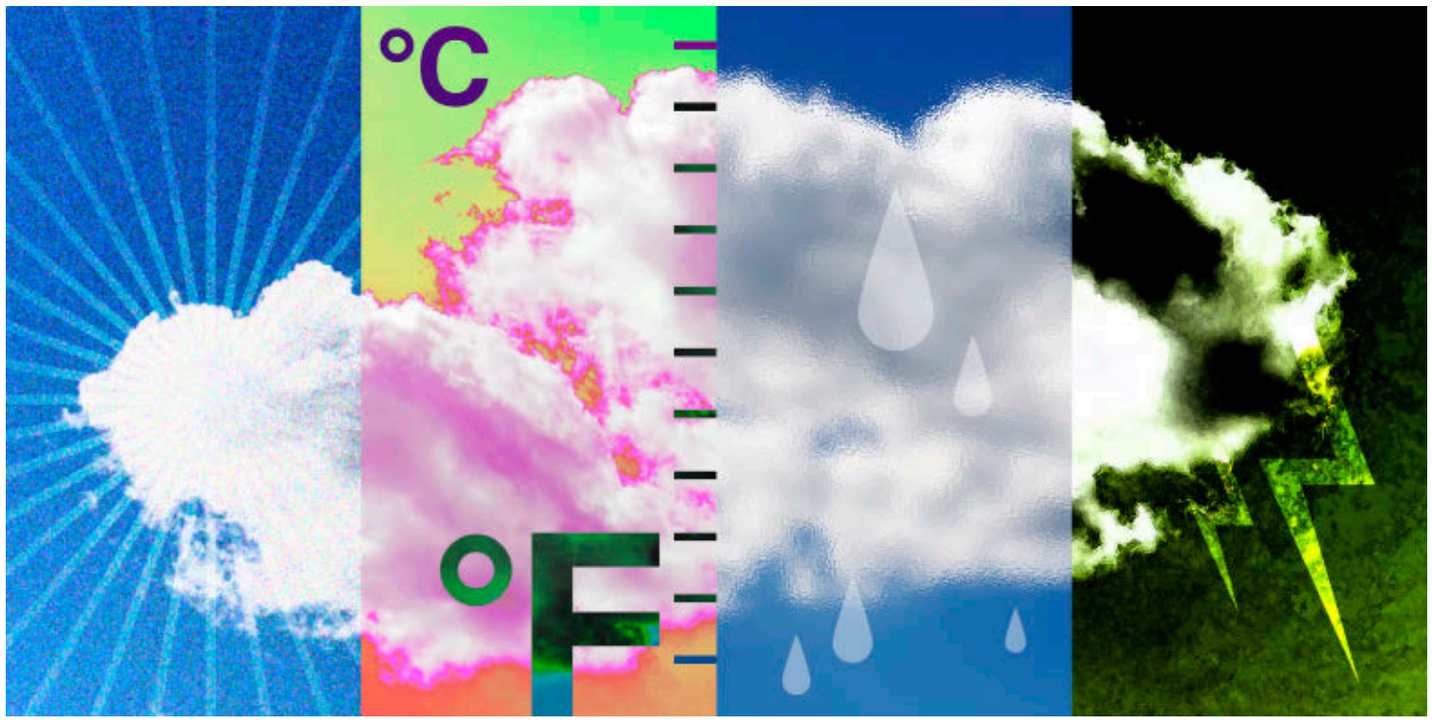
NightHacking Tour

Java Evangelist **Stephen Chin** recently wrapped up his three-week NightHacking Tour across Europe on a BMW motorcycle. He went to the Devoxx and JFall conferences and five Java user group events, and interviewed more than 30 NightHackers in 7 countries. Highlights of the tour included hacking events in London, England; Munich, Germany; Turin, Italy; and Lyon and Paris, France; and a behind-the-scenes tour at Aldebaran Robotics. Watch all of the interviews.



Oracle's Yolande Poirier chats with Stephen Chin.

HITACHI DEMONSTRATES SMART SENSOR INTEGRATION WITH THE CLOUD



At JavaOne 2012, Hitachi's OSGi SuperJ Engine Framework was deployed to integrate data collection from smart sensors (scattered throughout the conference venue) with the cloud. The framework applies Oracle Java SE Embedded to provide the sensors with decision-making intelligence and instrument diagnostics. The sensors collected more than 200,000 light, temperature, humidity, and energy measurements in a two-day period.

The raw measurements were preprocessed by the Java Embedded SuperJ Engine application (which ran on a GlobalScale Technologies Mirabox device). The preprocessed results were transferred to a SeeControl cloud-based global data analysis application, which collated and processed the data into graphic displays and made them available on the Web.

The demonstration was an illustration of the emerging machine-to-machine (M2M) Internet of Things that was a key focus of the very first Java Embedded @ JavaOne conference.

ART BY I-HUA CHEN

M2M

Pipelines on the Rise

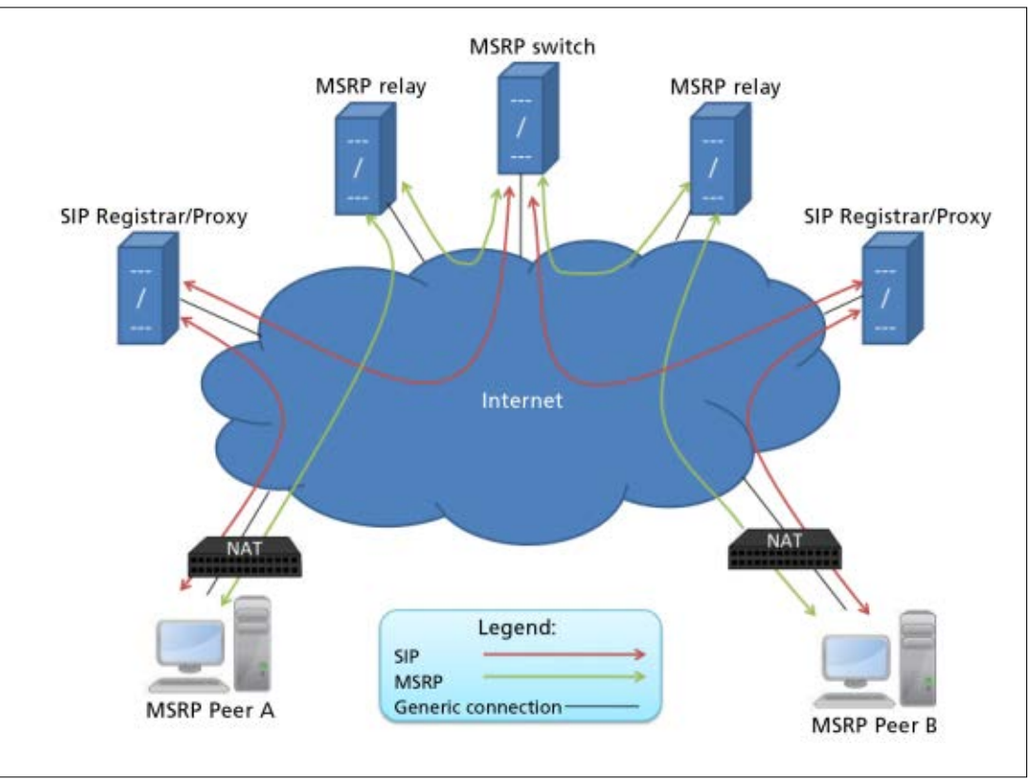
Oracle commissioned a recent survey by machine-to-machine (M2M) market research firm Beecham Research, which revealed that development of automated M2M data processing pipelines, in which information gathered by remote sensors in embedded devices is transferred to data centers for further analysis, is rapidly increasing. The global survey included end users, major product original-equipment manufacturers (OEMs), systems integrators, mobile network operators, and M2M platform providers.

Most survey respondents viewed M2M data pipelines as being critically important. A full **75 percent** said their M2M projects are utilized to create new services. However, **85 percent** noted that the large volumes of data that are passed from the remote embedded devices to the data center create new performance issues. Efficiently filtering and processing the data to enable its use for making timely decisions was cited as another difficulty.

View the complete results.

FEATURED JAVA.NET PROJECT

MSRP



This diagram displays the MSRP concept.

Java.net's Message Session Relay Protocol (MSRP) project (RFC 4975) started out as a 2008 Google Summer of Code project. Founder **João Antunes** wanted to create a library to provide Jitsi, an open source Java video chat application, with file transfer capability via Session Initiation Protocol (SIP, RFC 3261). Antunes preferred creating a full library rather than a Jitsi client, so the code could be used in any Java SIP application that requires file transfer capability.

In 2010, **Tom Uijldert**, of ContactMakers, took over MSRP project leadership duties. He is currently the primary developer, and introduced ContactMakers to the features and capabilities the MSRP library provides. "The MSRP library provides an ideal means for integrating VoIP [Voice over IP] and chat into a single platform, due to its SIP connectivity and standards-based programming model," he says. As a result, ContactMakers is currently integrating the library into its call center platform, and they have also become a corporate sponsor of the MSRP project.

The project has come a long way in a very short time. In late November 2012, Version 1.0.4 was released. Antunes says, "Today, the project implements all of the requirements of an MSRP peer." "It's now ready to be deployed in the real world," adds Uijldert.

What's coming up? Uijldert lists several potential enhancements, including integration with the Mobicents JAIN SLEE platform, and the development of a C# version of the MSRP library. He says that the biggest challenges for this project will be the addition of secure communication, and possible support of relaying (RFC 4976). And Uijldert is actively seeking assistance with all this work: "Anyone interested in helping out with the additional functionality is hereby invited to contact us," he says.

Reflecting on open source and the MSRP project, Antunes says, "The open source spirit is great! With open source, you are making the world go forward. Plus, working on an open source project is good for your résumé. Just sprinkle your code with sufficient comments, and apply good coding conventions. That is the best way for your project to invite third-party contributions."

JAVA USER GROUP PROFILE

PHILADELPHIA AREA JAVA USERS' GROUP

The Philadelphia Area Java Users' Group (PhillyJUG) was founded in 2000 by **Dave Fecak**, a technical recruiter, and **Dr. John Lewis**, a former professor at Villanova University and coauthor of a popular early Java textbook. About 30 developers attended the first meeting in March 2000.

In 2005, the group was named to Sun's Top 25 program and then to the expanded Top 50 program. These programs recognized the leading JUGs. "That recognition and some good connections have helped us to attract some very high-profile industry names, such as **Rod Johnson** (Spring), **Gavin King** (Hibernate), **Marc Fleury** (JBoss), **Neal Ford** (ThoughtWorks), and **Eric S. Raymond** ('The Cathedral and the Bazaar')," says Fecak.

Today, more than 1,300 members subscribe to the group's e-mail list. About eight meetings are held annually, and these regularly draw more than 100 attendees, and in the 200 range for the most-popular speakers.

Looking ahead, "We are currently transitioning from a Java language focus to a JVM [Java Virtual Machine] focus, which the membership feels will keep us current with industry trends," says Fecak.

Fecak has thought a great deal about the JUG leader role. "I would encourage JUG leaders to always be thinking of how to help their membership. When [vendors] approach me about sponsoring a meeting, I always ask what I can get for my members. Sponsors want to reach audiences, and many sponsors understand that they are building corporate goodwill in the community for their brand." However, he notes that PhillyJUG does not allow pure product demonstrations or any sales activity.

OptionsCity Wins Chicago Innovation Award

What platform would you choose if your business required solving complex mathematical decision-making problems within milliseconds, where the correctness of those results directly and significantly affects your customers' financial reserves? OptionsCity Software, which develops electronic financial market trading solutions, chose Java. In October, the company's Freeway multiasset algorithmic trading platform won a Chicago Innovation Award. These awards honor innovative new products and services brought to market from the Chicago area.

OptionsCity Director of Client Technology **Freddy Guime** calls the win "a Cinderella story for Java in the financial industry." The OptionsCity development team "solved complex problems to achieve the performance needed for a submillisecond trading platform, which is necessary to compete in this space," he says.

Guime notes, "Even though there were naysayers when we chose Java as our platform, the decision allowed us to quickly take over this space, culminating with (what we consider) the first commercially viable Algo Trading Platform. And that includes the innovation of the Algo Store, which allows algorithm developers to be able to purchase 'components' of an algorithm."

Within six years, OptionsCity has gone from unknown to being one of the big players in the options trading platform space in Chicago and New York. Need reliable, time-critical mathematical algorithm performance in your startup product? Think Java.

End of Public Updates for Java SE 6

Have you updated to Java SE 7? Along with great features such as fork/join, NIO, and Project Coin, Java SE 7 is being updated and patched regularly. Java SE 7 has been out for more than a year and is ready to download.

The last publicly available release of JDK 6 is to be released in February 2013. After February 19, all new security updates, patches, and fixes for Java SE 6 and Java SE 5 will be available only through My Oracle Support and will thus require a commercial license with Oracle. It's important for developers and systems administrators either to make the transition over to Java SE 7 or to work with Oracle to get updates via the Java SE Support program.

In the event that you are not ready to migrate to Java SE 7, Oracle offers Java SE support for continued access to critical bug fixes and security fixes as well as general maintenance for JDK 6. Additionally, Oracle Java SE Advanced and Oracle Java SE Suite offer superior diagnostics and manageability tools that minimize the costs of deployment, monitoring, and maintenance of Java-based IT environments.

JAVA CHAMPION PROFILE

ADAM BIEN

Adam Bien *is a German entrepreneur, architect, programmer, author, and speaker. He became a Java Champion in January 2007.*

Java Magazine: Where did you grow up?

Bien: In a small town in Bavaria, Germany.

Java Magazine: How did you first become interested in computers and programming?

Bien: I was always interested in spaceships, aliens, UFOs, robots, and black holes. Computers were a natural progression. I had my first contact with a computer when I was eight years old.

Java Magazine: What was your first computer and your first programming language?

Bien: My first computer was a ZX Spectrum 128K. I got it when I was 12 and immediately started to program in Basic. However, I thought everything happened inside REM statements. My programs always interpreted without error, but nothing actually happened. I wondered about this. After getting the basics, I was proud to learn my first pattern: GOSUB instead of GOTO.

Java Magazine: What was your first programming job?

Bien: I had to optimize the wire protocol of an early Oracle database on a VAX. I didn't have a VMS OS, so I used Linux to compile the C++ code.

Java Magazine: What do you enjoy for fun and relaxation?

Bien: I try to enjoy everything. Life is too short. I consider Java programming as fun and relaxation. Someday, I would like to be able to read more non-IT books

and do more sports. Right now I have to catch the unchecked LackOfTimeException way of trying that—with an empty catch block, unfortunately.

Java Magazine: What happens on your typical day off from work?

Bien: I try to spend all my time with family—but there are really no days off. For freelancers, the day after a free day, you'll regret it.

Java Magazine: What "side effects" of your career do you enjoy the most?

Bien: With Java, you get endless opportunities to learn new things. Indeed, Java still feels to me like the new hot stuff, and this keeps my work exciting and absolutely not boring.

Java Magazine: Has being a Java Champion changed your daily life?

Bien: As Java Champion, I was "exposed" to JUG leaders, and I learned about the wealth of worldwide JUG events. I regularly speak at local JUG events when I travel on business. It has also been pleasant to meet lots of passionate, talented, and opinionated fellow Java Champions.

Java Magazine: What are you looking forward to in the coming years?

Bien: Lots of interesting things will happen in the near future for Java: JavaFX, Project Nashorn, lambdas, Java EE 7. And I cannot wait for Jigsaw. Java's future is bright.

PHOTOGRAPH BY PIOTR MALECKI/GETTY IMAGES

EVENTS

Embedded World *FEBRUARY 26–28, NUREMBERG, GERMANY*

The Embedded World Exhibition & Conference has been the gathering place of the international embedded community for more than a decade. The event focuses on all fields of embedded systems development. In addition to discussing hardware, software, and tools, embedded experts tackle the latest issues around embedded system design. Machine-to-machine is a key theme of the event. The conference also offers an extensive exhibition hall with more than 800 exhibitors.

PHOTOGRAPH BY GETTY IMAGES

FEBRUARY

FOSDEM

FEBRUARY 2–3
BRUSSELS, BELGIUM
FOSDEM is a free, noncommercial event that offers open source communities a place to meet, share ideas, and collaborate.

Jfokus 2013

FEBRUARY 4–6
STOCKHOLM, SWEDEN
Jfokus is the largest annual conference for everyone who works with Java in Sweden.

Java Israel

FEBRUARY 14
TEL AVIV, ISRAEL
Clojure, Grails, Groovy, Java Virtual Machine, JRuby, Ruby, and Scala are all on the agenda at this one-day Java event.

DevNexus

FEBRUARY 18–19
ATLANTA, GEORGIA
Presented by the Atlanta Java Users Group, this conference for Java developers offers seven tracks with three keynotes and more than 50 sessions.

Hadoop Innovation Summit

FEBRUARY 20–21
SAN DIEGO, CALIFORNIA
This event brings together data scientists, engineers, and architects pioneering Hadoop innovation for two days of keynote presentations, interactive workshops, and networking opportunities.

Mobile World Congress

FEBRUARY 25–28
BARCELONA, SPAIN
Explore the new mobile horizon at Mobile World Congress. This industry-leading event includes a thought-leadership conference with keynotes and panel discussions, a product and technology exhibition with 1,500 exhibitors, and an awards program.

MARCH

QCon London

MARCH 4–5 (TRAINING),
MARCH 6–8 (CONFERENCE)
LONDON, ENGLAND
QCon London is the seventh annual London enterprise software development conference designed for developers, team leads, architects, and project management.

33rd Degree Conference

MARCH 13–15
WARSAW, POLAND
This conference for Java masters features three days of Java technology talks. Birds-of-a-feather and unconference sessions allow informal discussions. In-depth workshops are also included.

CodeMotion

MARCH 24
MADRID, SPAIN
This event, by and for technology communities, gathers technicians, developers, and students. Organized by user groups for a variety of technologies, CodeMotion is a way for attendees to step out of their comfort zones and discuss innovation in coding practices.

EclipseCon North America

MARCH 25–28
BOSTON, MASSACHUSETTS
This conference focuses on sharing best practices, insights, case studies, and innovations in the Eclipse community and in software development.

Devoxx London

MARCH 26–27
LONDON, ENGLAND

Devoxx France

MARCH 27–29
PARIS, FRANCE
Following the success of the 11th Devoxx Java conference in November, Devoxx moves to the UK for the first time and France for the second time. The proximity of these back-to-back conferences allows speakers to present at both. At the Paris event, 75 percent of sessions are in French and 25 percent in English.

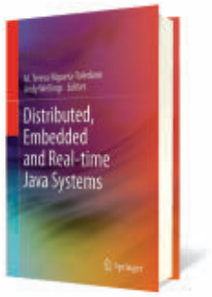
ORACLE AUTHOR PODCASTS

Our favorite Java authors give you the inside scoop on their books.

Benjamin Evans and **Martijn Verburg** discuss *The Well-Grounded Java Developer*.

Jim Weaver and **Stephen Chin** discuss *Pro JavaFX 2*.

JAVA BOOKS



DISTRIBUTED, EMBEDDED AND REAL-TIME JAVA SYSTEMS
M. Teresa Higuera-Toledano and Andy J. Wellings (Editors)
Springer (February 2012)
This book is aimed primarily at researchers in real-time embedded systems, particularly those who wish to understand the current state of the art in using Java in this domain. Much of the work in real-time distributed, embedded, and real-time Java has focused on the Real-Time Specification for Java as the underlying base technology, and consequently many of the chapters in this book address issues with, or solve problems using, this framework.

CONTINUOUS ENTERPRISE DEVELOPMENT IN JAVA (EARLY RELEASE)
By Andrew Lee Rubinger, Dan Allen, and Aslak Knutsen
O'Reilly Media (September 2012)
This practical book shows you how to perform continuous development, using a testing platform called Arquillian that the authors built with the JBoss community. This platform acts as the missing link between testing and development. The authors demonstrate how testing is the very foundation of development—essential for learning and critical for ensuring that code is consumable, complete, and correct. Find out how Arquillian helps you document your project in the form of test cases.

DEVOPS FOR DEVELOPERS
By Michael Hüttermann
Apress (September 2012)
DevOps for Developers delivers a practical, thorough introduction to approaches, processes, and tools to foster collaboration between software development and operations. Efforts of agile software development often end at the transition phase from development to operations. This book covers the delivery of software—"the last mile"—with lean practices for shipping the software to production and making it available to end users, together with the integration of operations with earlier project phases (elaboration, construction, and transition).

JAVA EE DEVELOPMENT WITH ECLIPSE
By Deepak Vohra
Packt (December 2012)
Java EE Development with Eclipse is a practical guide for using the most-commonly-used Java EE technologies and frameworks in the Eclipse integrated development environment (IDE). The book focuses on developing applications with some of these technologies using the project facets in Eclipse 3.7 and its enhancement Oracle Enterprise Pack for Eclipse 12c. It starts with a discussion of Enterprise JavaBeans (EJB) 3.0 database persistence with Oracle WebLogic Server and Oracle Database, Express Edition.

JCP Executive Series

Credit Suisse Discusses the Java Community Process

Susanne Cech Previtali and **Victor Grazi** share Credit Suisse's distinctive perspective. **BY JANICE J. HEISS**

We continue our series of interviews with distinguished members of the Executive Committee of the Java Community Process (JCP) with Credit Suisse, a financial services group of companies with headquarters in Switzerland and operations in more than 50 countries. Credit Suisse is the first nontechnology vendor on the Executive Committee of the JCP, elected in June 2010 and re-elected in November 2012. Credit Suisse is what Mike Milinkovich of the Eclipse Foundation calls a "Java user company" whose focus is to improve the Java platform in order to meet both its customers' and its own business needs. As such, the company brings to the JCP a unique and valued perspective.

PHOTOGRAPHY BY CHRIS LANE/GETTY IMAGES

Victor Grazi and Susanne Cech Previtali take a breather on Credit Suisse's rooftop deck in Manhattan.

We met with Credit Suisse's two representatives on the JCP, Dr. Susanne Cech Previtali and Victor Grazi. Cech Previtali, a member of the Infrastructure Architecture group at Credit Suisse, is the Java platform architect responsible for Credit Suisse's Java Application Platform (JAP) and the related construction guidelines for the development of large-scale Java EE applications. Grazi, who recently won the annual JCP award as Best Spec Lead for his work on JSR 354, Money and Currency API, is a Java Champion who works both in investment banking architecture and as a technical consultant and Java evangelist.

Java Magazine: Credit Suisse was recently re-elected to the Executive Committee of the JCP. While your position statement is available online, would you like to offer a general summary of your position going forward?

Credit Suisse: We are very happy about this great and encouraging result. We will continue to bring in our customer view in arguing for strong, open, and stable standards that can be implemented by any vendor or the open source community for a competitive market. We want to bring in our expertise in standardization and architecture governance and to continue our active participation in Expert Groups and as spec leads. We would like to aid in increasing university participation as spec leads and Expert Group members at the faculty level—for example, contributing to high-quality specifications; also, we think students should be included in making contributions at the development level—for example, as part of the Adopt-a-JSR initiative of the Java user groups [JUGs].

Java Magazine: Credit Suisse, the first nontechnology vendor on the JCP, has been characterized as a Java user company with the goal of improving Java in order to better meet your own and your customers' needs. This gives you a different perspective from a company that creates and sells software. Does your perspective clash with the perspectives of other JCP members? Does it offer an opportunity for other Executive Committee members to better understand how companies use Java?

Credit Suisse: Credit Suisse sees its role within the Java community in general, and the Executive Committee more specifically, as a representative customer interested in building mission-critical applications on strong, open, and stable standards to secure our investments. Credit Suisse previously contributed to the development of Java EE specifications through participation in many customer advisory boards, through statements of requirements for extensions to the core Java-related products in use, and through active

participation in JSRs.

The JCP Executive Committee is still dominated by vendors. Major customers of Java technologies as the end users were not integrated in the process and, as such, could not participate in guiding the development and evolution of Java technology. Credit Suisse stands in close relationship with the vendors. Our experiences and feedback with the concrete application of the Java technologies in, for example, our Java Application Platform are also interesting and relevant for the vendors.

Java Magazine: Why did you decide to join the JCP in 2010? How might your presence in the JCP help Credit Suisse?

Credit Suisse: Oracle nominated Credit Suisse to run for the Executive Committee seat of the Java EE/SE Executive Committee. Credit Suisse was elected by the JCP community with 77 percent for a ratified seat starting in June 2010, and was recently confirmed with 85 percent. Credit Suisse was among the first financial institutes to embrace Web and Java technologies. It was the first Swiss bank to provide a Web-based payments solution for its customers back in 1997, and its 1999 product, YouTrade, was one of the first client trading applications on the market. Credit Suisse developed an internal standard based on Java EE called the Java Application Platform [JAP], which helped industrialize development and operations of our Java EE applications. JAP also established many of the platform-as-a-service [PaaS] concepts in 2004 that now contribute to more than US$40 million of avoided costs per year. JAP featured one of the first large-scale SOA implementations, included the first ubiquitous PKI [public key infrastructure], and reduced the server footprint through consolidation on shared servers. In addition, it pioneered the adoption of numerous Java EE technologies including transaction monitors, JSF [JavaServer Faces], and portal.

Today, Credit Suisse invests significant amounts per year in the development of new and maintenance of existing Java EE applications, and owns more than 30 million lines of Java code. Going forward, Credit Suisse intends to steadily increase that investment as we move more workload from the mainframe back end to our Java EE platform.

Java Magazine: Does your vision of Java EE differ from that of other JCP Executive Committee members?

Credit Suisse: Our general vision of Java EE is focused on the support of cloud features and modularization. The standardization of PaaS features such as provisioning, multitenancy, elasticity, and the deployment of applications in the cloud are crucial for developers and operation. The modularization of the Java SE platform and the subsequent possibility of better capturing dependencies and versions across modules are important for large-scale applications. These are major changes with major impact to the Java platform and thus have been moved from Java EE 7 to Java EE 8. The current vision is common, although we may have differences about roadmaps and timelines.

As JCP representatives, Grazi and Cech Previtali offer a customer view in arguing for strong, open, and stable standards.

Java Magazine: How has the JCP changed since Oracle acquired Sun Microsystems?

Credit Suisse: The most important change is the JCP.next initiative, which is a commitment to greater transparency, participation, and openness. JCP .next.1, JSR 348, Towards a New Version of the Java Community Process, laid the foundation for greater transparency by requiring, rather than suggesting, that all development is done on open mailing lists and issue trackers. Furthermore, the recruiting process for Expert Group members will be publicly viewable, with transparent disclo-

CHANGE IS GOOD

"The most important change is the JCP.next initiative, which is a commitment to greater transparency, participation, and openness."

THE CHALLENGE

"Trying to build consensus is difficult because people have individual ideas about best practices… uncovering all of the use cases and coming up with a robust implementation is the overriding goal."

sure. TCK [Technology Compatibility Kit] testing process results will be investigated; currently, the public is rarely aware of the results of the TCK testing process. All of these developments are designed to result in a more public, open, accessible, and transparent JCP.

JCP.next.2, JSR 355, JCP Executive Committee Merge, addresses the merging of the two Executive Committees (Java SE/EE and Java ME) and thus lays the foundation for one platform.

JCP.next.3, JSR 358, A Major Revision of the Java Community Process, revises the JSPA [Java Specification Participation Agreement]. This JSR will make changes to the JSPA, the process document, and the Executive Committee's standing rules with the goals of further improving the organization's processes, correcting problems that have become apparent over recent years, and clarifying language to reduce ambiguity.

Java Magazine: What structural changes would you like to see in the JCP?

Credit Suisse: Openness, transparent licensing models, ease of participation, broad participation of the Java community, and awareness of the importance of technology governance.

Java Magazine: Has Oracle delivered on the promise of increased transparency and openness in the JCP?

Credit Suisse: Yes. Oracle is a member of the Executive Committee and has agreed and committed to JCP.next.1 and JCP.next.2 and collaborates on JCP.next.3.

Java Magazine: How could Oracle have increased participation in the JCP?

Credit Suisse: The inclusion of the JUGs in the Executive Committee is a great success, because it raises the awareness at the right level.

Java Magazine: Are there ways that the JCP could better serve the Java community?

Credit Suisse: The JCP drives the evolution of a programming language and its ecosystems and has become a very important asset for the Java community. By relating the needs of the users with those of the technology providers, the JCP can ensure that Java continues to be one of the top programming languages. We should also not forget about the next generation, and make sure that faculty and students continue to see the value of Java.

Java Magazine: What is your biggest complaint about the JCP?

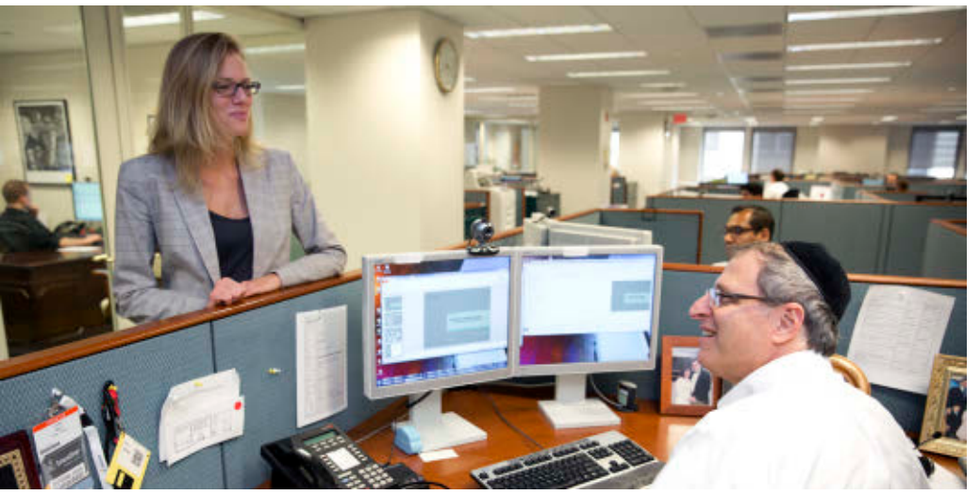
Credit Suisse: The Java community is great and very active, but not all users of Java technology are aware of the importance and impact of technology governance. The JSRs are developed by the JCP community, and so perhaps we can do better at promoting ourselves to include more people and companies to engage in the evolution of Java technology.

Java Magazine: There seems to be a risk that members of Expert Groups in the JCP can be subtly influenced by their own use cases, which suggests that there is a challenge in remaining objective about what a specification needs and does not need. How do you filter out any biases you might have related to the influence of your own use cases?

Credit Suisse: Trying to build consensus is difficult because people have individual ideas about best practices, which at times may conflict. There is some ego involved in this, too, but uncovering all of the use cases and coming up with a robust implementation is the overriding goal. All spec leads have similar challenges, including finding the time while still managing their day jobs. Also, even when you find the time, it is difficult to get the other members involved, because they also have their own day jobs.

Java Magazine: Do you have any suggested improvements for Java.net as it relates to leading a JSR group?

Credit Suisse: Java.net is the JSR spec lead's best friend. It provides a wiki, source control, a mail list, archives, really everything you need to manage a JSR. We are very impressed with the fast feedback that the JCP provides for support issues. It would be great if some performance-based incentives could be provided to the Expert



Developers won't need to reinvent the wheel in creating financial applications, say Cech Previtali and Grazi, as a result of JSR 354.

Groups—perhaps some awards for most valuable contributor.

Java Magazine: Tell us some potential improvements for creators and users of financial applications that are in the works as a result of JSR 354. How will their work be made easier?

Credit Suisse: For one thing, developers won't need to reinvent the wheel when creating financial applications. A common error is to use a float or double to represent financial values. However, these formats are not designed for accuracy, and can easily produce rounding errors in the *n*th decimal place. Because accountants don't appreciate rounding errors, a lot of thought needs to go into how to make these precise. In addition, JSR 354 can help with formatting. In the US and Europe, most countries use very standard formats to express numbers, but countries such as India use lakhs and crores, which are specified quite differently from anything currently available in the Java formatters. This will improve with JSR 354.

Also, rounding rules are important. In the US, it's standard to round up for 5 or more and to round down from 1 to 4. But different countries have more-exotic rules for rounding.

Java Magazine: In walking the fine line between respecting standards and encouraging innovation, does the JCP err too far in one direction?

Credit Suisse: As customers, we count on the standardization of technology because it gives us flexibility in choosing products and, thus, reducing vendor lock-in. Some people argue that there should be at least two implementations available before the standardization. But what about migration paths of the existing implementation? What about migration paths for the customers who already have used that existing implementation? This could result in very weak standards, costly migrations, or both. Standardizing too early is problematic, because immature technology does not provide enough useful "material." Standardizing too late is costly and probably causes major changes for customers who switch to a standard technology. It's a delicate balance. `</article>`

PROMOTING CHOICE

"We count on the standardization of technology because it gives us flexibility in choosing products and, thus, reducing vendor lock-in."

Janice J. Heiss is the Java acquisitions editor at Oracle and a technology editor at *Java Magazine*.

LEARN MORE

- Java Community Process
- Credit Suisse

Globalize Your Business

Melissa Data can help you globalize your applications as you expand operations to other countries or reach new customers in emerging markets. As a world leading data quality vendor, we offer solutions to verify, correct and standardize addresses in over 240 countries. Eliminate returns, cut postage expenses, prevent fraud and keep your customers happy by verifying their address before you send a package.

- Reduce address correction fees – save up to $10 per package
- Efficiently validate and correct addresses every time you ship
- Maintain high customer satisfaction

Accurate data. Delivered.

- ✔ Address Verification
- ✔ ID Verification
- ✔ Email Verfication
- ✔ GeoCoding
- ✔ IP Location
- ✔ Name Parsing
- ✔ Phone Verification
- ✔ Record Matching

www.MelissaData.com/global
or call 1-800-MELISSA (634-4772)

MELISSA DATA®
Your Partner in Data Quality

On premise or remote APIs available

EMBEDDED **EVERYWHERE**

Terrence Barr on Java and the Internet of Things **BY STEVE MELOAN**

While number estimates might vary, it's clear that within this decade we will see tens of billions of new embedded computational devices entering our everyday lives—including connected vehicles, smart appliances, smart vending machines, smart meters, medical sensors, industrial controllers, and more. And Java's write once, run anywhere technology is positioned for this coming wave of diverse embedded

ART BY NICHOLAS PAVKOVIC, PHOTOGRAPHY BY TON HENDRIKS

SUPER VIRTUALITY

"Because Java, by its nature, is a virtual machine [VM] environment and a virtual application platform, you can start developing and testing large parts of your application on a virtual environment—on the desktop, for example—before the embedded hardware is available."

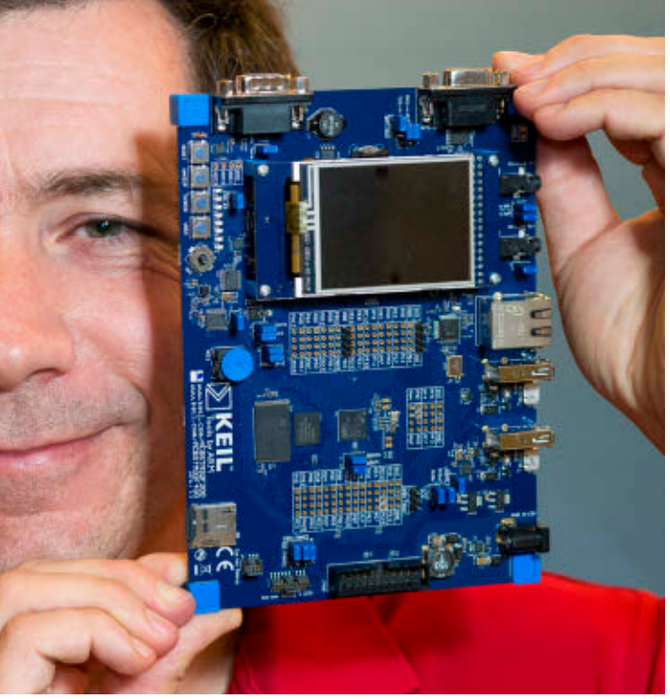
devices—offering an underlying platform and infrastructure that include open standards, security, reliability, scalability, sophisticated toolsets, and a 9-million-plus developer community.

Java Magazine *sat down with Terrence Barr, senior technologist and product manager at Oracle, to explore the Internet of Things, and Java's role in a technology transition that some are calling a third IT revolution.*

Java Magazine: What is the advantage of using Java in the embedded space?
Barr: The embedded space today is very fragmented. You have to piece together the entire stack—the runtime, the OS , the tools, the languages, the protocols, and the connectivity. We strongly believe that until we have a standards-based horizontal platform, the M2M [machine-to-machine] market won't take off, because too much time is spent reinventing the wheel and struggling with fragmentation rather than building solutions.

Java is a platform and a technology that, by design, abstracts from the underlying hardware and OS, and provides a rich application environment, from very small devices to enterprise-class servers. Java is uniquely positioned to address many of these challenges because, essentially, we solved these problems 10 or 15 years ago. We solved transitions from 8- to 16- to 32-bit that the embedded space is facing today. We have a proven and secure runtime environment. We have infield updatability of application components. And Java already has a technology ecosystem with 9 million developers. They might not currently be embedded developers, but they're very familiar with the programming language and the toolset.

Java Magazine: So Java not only facilitates development in the embedded space but also acts as a *catalyst* toward embedded industry growth?
Barr: Exactly. An Oracle technology partner recently drew a parallel between the current embedded space and the early cell phone space. The whole ecosystem was held back by the fact that there was no standard platform that people could build upon. But as soon as fairly predictable phone platforms became accessible, you had this explosion of applications, accessories, hardware, and services that built upon the platforms. We see Java in the embedded space playing a very similar role, but with even more focus on creating an open, standardized technology infrastructure.
Java Magazine: What are the time-to-market advantages of developing embedded applications via desktop emulation?
Barr: Because Java, by its nature, is a virtual machine [VM] environment and a virtual application platform, you can start developing and testing large parts of your application on a virtual environment—on the desktop, for example—before the embedded hardware is available. So you can parallelize the embedded hardware development and the embedded software development, which potentially shaves many months off of your development cycle.
Java Magazine: What are the issues regarding pushing down new logic to



Barr shows off an application running on a Keil board at Devoxx.

embedded devices—for updates and delivery of new services?

Barr: Suppose you have millions of smart meters in the field, running the software, collecting the data, and sending the data to the back end. But at some point, the utility laws and regulations change, and you need to update the algorithms on these meters. It's not plausible to send people into the field to manually update millions of devices. Embedded hardware is getting more and more powerful, and at the same time connectivity options are increasing, which allows for wireless updates. Java offers the ability to update in the field without affecting the integrity of the system. Because Java is a virtual environment and runs in a sandbox model, it helps guarantee the security and integrity of the system after the update. Also, over time, you might deploy, say, 20 variations of hardware devices. Without Java, you'd have to update 20 types of native code. With Java, you write the application once, and you push it to multiple devices without concern for hardware dependency.

Whether it's utility meters, automobiles, medical devices, or vending machines, all have similar requirements—you need to execute applications, and you need connectivity, security, a way to manage the device, a way to update the device, and a way for the device to connect back to the server. Delivering embedded Java as a horizontal and integrative solution provides 80 percent of your underlying infrastructure. You can then build 20 percent on top of it, adding your particular industry value. You get to market much quicker and at a lower price point.

A RICH FOUNDATION

"Delivering embedded Java as a horizontal and integrative solution provides 80 percent of your underlying infrastructure.... You get to market much quicker and at a lower price point."

Java Magazine: What are the development advantages of working with embedded Java?

Barr: As I mentioned earlier, with traditional native embedded development, you have to pick and choose the various components, integrate them yourself, and deal with a range of tools and methodologies. By comparison, the Java runtime is a complete development stack that is tested, is based on standards, and uses standardized APIs that you're already familiar with. So it's very easy to get started and write application and business logic. The new Oracle Java ME Embedded 3.2 not only offers a variety of standard APIs—things like XML processing, location-based services, and file access. We've also added APIs that are specific to the embedded space, that allow you to deliver a solution even more quickly. You're able to remotely provision and start and stop applications. You have a logging API that allows you to log locally as well as remotely, for remote-monitoring purposes. And you also have the AccessPoint API, which allows the application to intelligently switch between multiple communications channels depending on data rates and bandwidth requirements.

And finally, you can use your same familiar development tools, such as NetBeans and Eclipse. And you can do live, on-device source-level debugging using these IDEs. For most Java developers, that's nothing new. But for native embedded developers, this is heaven. For them, debugging and testing is typically a very painful process.

Java Magazine: What specific hardware would you recommend for developers who want to get their feet wet with embedded Java?

Barr: If you have experience in the embedded space, you know that there

PORT YOUR EXPERTISE

"One of the big values of having Java in the embedded space is the ability to bring the expertise, the tools, and the code from one platform to another—the inherent ability to apply your expertise across this whole range of devices, from very small to very large."

are any number of permutations of processors, I/O, board designs, interfaces, and peripherals—it's much more fragmented than the desktop space. So we had to pick a specific set of hardware to start with, where we could enable developers to get a system up and running, to experiment with some of the features that we support. So what we chose is the Keil MCBSTM32F200 (ARM Cortex-M3)—a ready-to-run evaluation platform. You can buy it online, and once you have it in your hands, you'll soon be able to download the Oracle Java ME Embedded 3.2 runtime binary. That will come with instructions, and you'll flash the runtime onto the board, hit the reset button, and you'll have a complete, ready-to-run Java runtime environment. You'll hook up your IDE—NetBeans or Eclipse—and you'll be ready to write embedded Java applications. In the meantime, we offer a Windows-based device emulation environment.

The Keil board is mainly for evaluation purposes. If you want a deployment-ready device, we can point you to our partners, such as Cinterion and Qualcomm. They will be making available various types of integrated, wireless module devices and platforms that come with the Oracle Java ME Embedded 3.2 runtime. The devices have an integrated cellular modem, they have RAM and ROM on the module—they're basically very powerful small wireless computers. So you can also use these to get started with embedded development and for your actual device deployments.

Java Magazine: The timeline for Oracle's embedded offerings details a 2014 JVM [Java Virtual Machine] convergence for Oracle Java SE Embedded 8 and Oracle Java ME Embedded 8. What will this mean for embedded developers and partners?

Barr: One of the big values of having Java in the embedded space is the ability to bring the expertise, the tools, and the code from one platform to another—the inherent ability to apply your expertise across this whole range of devices, from very small to very large. So about 12 to 18 months ago, Oracle embarked on a plan to converge and make more aligned the two relevant Java platforms, Java SE and Java ME.

Historically, Java ME was somewhat the smaller sibling of Java SE. But it had different features and APIs, which made it difficult to take your code and expertise and switch platforms. This strategy of aligning the two platforms will make that process much easier—with similar language features, similar API features, and similar runtime features. To that effect, Oracle filed two new JSRs, JSR 360 and JSR 361, in October 2012. In 2013 and 2014, we will continue to execute on this strategy to reshape the Java platform and the Java products. The goal is that eventually developers see Java as one continuum, from the very small device up to the very large systems. So you can apply your same skills, tools, and code across this continuum.

Barr talks about embedded while onsite at Devoxx.

Barr talks with Stephen Chin about embedded Java and demos a smart solar tracking system.

You'll choose the runtime based on your device properties, but from a developer perspective it will look very smooth and homogeneous.

Java Magazine: For those outside the embedded world, a Java ME configuration provides the basic set of libraries and VM capabilities for a range of devices, while a Java ME profile adds a set of APIs that enable a more specific range of use cases. How will the Java SE and Java ME convergence affect the current CDC [Connected Device Configuration] and CLDC [Connected Limited Device Configuration]?

Barr: The CDC traditionally occupies the midrange embedded space and has been very successful in vertical product spaces—for example, every Blu-ray player ships with a CDC runtime. But we want to map the entire embedded space. So CDC will basically become a configuration of Oracle Java SE Embedded, and Oracle Java ME Embedded will cover everything else. As a developer, you'll basically look at your problem domain and your device requirements, and you'll just say, "OK, does it fit a Java ME Embedded environment, or do I need Java SE Embedded?" The two platforms will be aligned, so from a development perspective the experience will be very similar.

LOCAL INTELLIGENCE

"What you want to do is use the intelligence of the local device to program-in business logic."

Java Magazine: It's difficult to overestimate the likely billions of connected smart devices in this era of the Internet of Things. At the back end, this will bring about a data explosion. How is Oracle positioning itself for this data tidal wave on the enterprise?

Barr: It's clear that these devices will be generating huge amounts of data—after all, the data is where their value is. If this anticipated data wave is left unchecked, it could overwhelm the internet infrastructure. Many cellular networks today already might be overwhelmed in handling the data from smartphones, so how are we going to handle data from billions of additional connected devices?

What you need to do is to turn large amounts of data into small amounts of valuable information. If you have a sensor that's monitoring the temperature in a truck, you don't want that device to just blindly send a temperature reading to a back-end server every second. That's not useful information, and it generates enormous quantities of data (and associated costs) if done on a large scale. And then on the back end, what do you do with that data? It has almost no value.

What you want to do is use the intelligence of the local device to program-in business logic. If you're transporting a certain type of product that has specific temperature require-

Oracle's Embedded Products

Oracle Java SE Embedded is based on Java SE but optimized for midrange to high-end embedded systems—devices having 32 MB or more of RAM (without graphics).

Oracle Java ME Embedded Client is based on the Java ME Connected Device Configuration (CDC), a subset of the Java SE platform optimized for low-to-midrange embedded systems—devices having 8 MB or more of RAM (without graphics).

Oracle Java ME Embedded 3.2 is an optimized runtime stack based on the Java ME Connected Limited Device Configuration (CLDC), intended for small, headless (no graphics/UI) embedded devices having 130 KB or more of RAM.

Oracle Java Embedded Suite 7.0 provides an enterprise-like suite of technologies for the Oracle Java SE Embedded 7 runtime—Java DB relational database; GlassFish Servlet-based application server; and Oracle's Jersey implementation of JSR 311, the Java API for RESTful Web Services (JAX-RS).

ments, you need to send an alert only if the temperature falls outside of that range. The embedded device processes the data locally, and the back end sees a vastly reduced level of transmitted data, in the form of actual useful information such as alerts.

But you're not just connecting those billions of embedded devices directly to the back end, because both the flow of data and the management of these devices would be difficult to handle. What you typically need is a *concentrator* tier and device, which sits between the edge client and the back end. The concentrator takes the data stream from these devices and filters it, extracting in real time the business-critical information (called *complex event processing*), and then sends it on to the enterprise at the back end. The concentrator can also act as a proxy to manage devices on behalf of the back-end tier.

BIG DATA CHALLENGE
"Many cellular networks today already might be overwhelmed in handling the data from smartphones, so how are we going to handle data from billions of additional connected devices?"

Java brings great value to all of these processing tiers. Depending on the given solution, you can choose where and how to implement filtering and business logic. And with Java, it's always the same language, the same programming model, and the same tools and expertise. That's a huge value over having different languages and different programming methodologies spread out across your solution.

Java Magazine: How does big data coming from embedded devices affect database product solutions?

Barr: NoSQL databases are frequently being used on these back-end systems, because they're more flexible in dealing with this kind of heterogeneous device data. But, in many cases, you also want to mesh together relational and nonrelational data. You could take temperature sensor data and location data from your devices, for example, and mesh that on the back end with mapping data to create more-valuable information.

Java Magazine: This year at JavaOne 2012, Oracle presented an entire subconference called Java Embedded @ JavaOne. What was the takeaway experience from the conference?

Barr: Oracle is dead serious about the M2M space and embedded Java. Oracle recognizes that this is a huge opportunity—for Oracle, for Oracle's partners, and for the entire Java ecosystem.

During the conference, I spoke to a lot of partners and developers, and the reaction was uniformly positive. They said, "We get it. This makes sense to us, why Java should be in the embedded space." Now we're busy fielding all sorts of requests from developers and potential partners who want to open up a dialogue and understand where we're going, how they can add value, and how they can participate. People are excited about the possibilities out there. For developers, I believe this is a huge opportunity. For Java, the best is yet to come. `</article>`

Steve Meloan is a former C/UNIX software developer who has covered the Web and the internet for such publications as *Wired, Rolling Stone, Playboy, SF Weekly,* and the *San Francisco Examiner*. He recently published a science-adventure novel, *The Shroud,* and regularly contributes to *The Huffington Post*.

LEARN MORE

- Java Embedded
- Java Embedded downloads
- Oracle Java ME Embedded 3.2 FAQ
- Terrence Barr's blog

(embedded)

TOP TEN REASONS FOR USING JAVA IN EMBEDDED APPS

Learn why Java is the best language for the Internet of Things.

BY SIMON RITTER

For many years now, we've been promised the Internet of Things—with everything from lightbulbs to cars all networked together with built-in "intelligence." The combination of Moore's Law and economies of scale is finally making this a reality. For example, Philips recently announced that it would be selling internet-enabled lightbulbs through the Apple store. For these networked, programmable devices to be really useful, they need applications to collect, process, and transmit data. Traditionally, code for embedded systems has been written in assembler or C, but for the rapid explosion of programmable, connected devices, we need something better: Java. What makes Java the best language for embedded devices? Here are ten great reasons (with thanks to Roger Brinkley's blog entry for inspiration):

ART BY NICHOLAS PAVKOVIC, PHOTOGRAPHY BY BOB ADLER

1 WE ALREADY HAVE A WHEEL; LET'S NOT KEEP INVENTING NEW ONES.

Java SE 7 has almost 4,000 standard class libraries covering everything from collections through concurrency to networking. Having this enormous collection of standard APIs means significantly less time spent rewriting standard functionality for embedded applications. While smaller-footprint parts of the Java platform, such as Oracle Java ME Embedded, don't have as many standard classes by default, they still have a wealth of functionality available. Many of the Oracle Java ME Embedded APIs are targeted at the needs of embedded systems: for example, the device API provides standardized ways of working with low-level protocols such as Serial Peripheral Interface (SPI) and Inter-Integrated Circuit (I2C). However, Java's API power doesn't stop with the classes available as part of the platform. Because Java is such a popular language, developers have created libraries that address almost anything you can think of: sensors, serial port communication, and a ZigBee API, to name a few. Many of these libraries are open sourced and can easily be integrated into new embedded applications.

ONE FOR ALL

Porting code in Java is a non-event. As long as none of the APIs used by the application have changed, it's simply a matter of redeploying the existing class or JAR files.

2 "SEGMENTATION FAULT" IS NEVER GOOD, SO WE'LL JUST AVOID IT.

One of Java's strengths, and one of the things that has made it so popular as a programming language, is how robust application code is. Languages such as C and C++ use explicit pointers to reference memory. In Java, all object references are implicit pointers that cannot be manipulated by application code. This avoids potential problems, such as buffer overruns and memory access violations, through incorrect pointer calculations. Situations such as these can cause an application to stop abruptly. In embedded systems, these types of errors can be much harder to track down because often there is no device such as a screen where error messages can be displayed.

3 ONE PLATFORM RUNS THEM ALL.

Unlike desktop machines or even servers, embedded systems vary widely in terms of their architecture, resources, and OS capabilities. When embedded systems are updated, there's also a good chance that the processor and OS will change significantly from the previous release of the product. Migrating applications written in assembler or C to a new platform can be time consuming, costly, and error prone. From the very beginning, Java has been write once, run anywhere. Porting code in Java is a non-event. As long as none of the APIs used by the application have changed, it's simply a matter of redeploying the existing class or Java archive (JAR) files. When moving to a newer (and therefore better-performing) version of Java, all that is required is a simple recompile.

4 LET THE VIRTUAL MACHINE TAKE CARE OF THE MUNDANE STUFF.

The Java Virtual Machine (JVM) provides a wealth of benefits to embedded developers that they don't get with natively compiled code.

Memory management is all handled automatically by the JVM; memory is allocated by instantiating an object, rather than having to use explicit calls to library functions such as malloc. Developers don't have to keep track of object references and explicitly deallocate memory either; the garbage collector handles all this. This substantially reduces the potential for memory leaks, which can have a far greater impact on embedded systems where applications might need to run for very long periods in memory-constrained

DON'T GO NATIVE

Some people wrongly think that by using a VM rather than native instructions, performance will be compromised.

environments without needing to be restarted.

Concurrency support for applications is also handled by the JVM. The ability to create and synchronize different threads of execution has been built in from the very beginning with Java. Even if the embedded platform on which the JVM is deployed does not support multithreading directly (something that's increasingly rare these days), it's still possible to emulate the functionality through the concept of green threads.

Some people wrongly think that by using a virtual machine (VM) rather than native instructions, performance will be compromised. With nearly 20 years of development in the JVM, these issues are no longer a significant factor. In certain situations, having precise knowledge, which is available only as the application is running, can lead to *better* performance than the performance of natively compiled code.

5 PICK AN EMBEDDED PLATFORM—ANY EMBEDDED PLATFORM.

Embedded systems are generally designed to solve a specific problem, whether it is how to provide in-car entertainment or how to monitor the pH level of part of an industrial process. As such, each system will have hardware tailored to the solution being developed, which gives designers far greater choice than when creating a new desktop or laptop system.

Many embedded chip manufacturers, such as Freescale and Broadcom, create processors and systems on a chip (SoCs) using architectures from companies such as ARM. Although this makes certain aspects of the different processors and SoCs standard, there are often small differences in terms of aspects such as floating-point processing, bus architectures, and so on. Java abstracts these differences away from the developers, making their lives significantly easier.

Reference implementations of the Oracle Java Embedded Client are available for Intel x86 and MIPS as well as ARM v5, v6, and v7 architectures running Linux (the most popular OS for embedded devices). Oracle Java SE Embedded is available for Intel x86 and IBM Power e500v2 and e600 systems as well as ARM v5, v6, and v7 architectures, again running Linux. This gives embedded systems designers plenty of choice about which platform to use.

6 THINGS HAVE CHANGED SINCE 1995.

Java was originally developed in the early 1990s to allow applications to be written for the Star7 PDA, which was an embedded device. When Java was first launched as a general-purpose computing platform in 1995, the average personal computer had 8 MB of RAM and a processor running at less than 200 MHz. Even today, the full Java SE runtime requires only a minimum of 64 MB of RAM to run on Windows XP. Typical low-end embedded systems now match or exceed these specifications.

Java was developed from the very beginning to work in resource-constrained environments, making it ideally suited to the needs of embedded systems today. Oracle Java ME Embedded will run in as little as 350 KB of ROM and 130 KB of RAM (look carefully, that's *kilobytes*, not *megabytes*), growing to 1.5 MB of ROM with 700 KB of RAM for the full, standard configuration. Oracle Java SE Embedded is designed for more-powerful systems, but still requires only 39 MB of ROM and 32 MB of RAM.

Having to deal with lower clock speeds also means that the Java platform works well when you want to maximize the time between charges for battery-powered devices. When resources are tight, Java can still get the job done.

Java Embedded: Start Developing

Tori Wieldt, Java Magazine *technology editor, caught up with Kevin Smith of Oracle Technical Business Development and got the latest on Oracle Java ME Embedded.*

Java Magazine: What's new for partners in the Java ME embedded space?

Smith: We announced Oracle Java ME Embedded at JavaOne San Francisco 2012. It's pretty exciting because it represents a new Java platform that addresses the needs of the machine-to-machine [M2M] market and the Internet of Things. We have been working closely with Cinterion and Qualcomm on this. Oracle Java ME Embedded contains JSR 228, Information Module Profile—Next Generation. The Java ME team has been working for many years connecting people to the internet with PCs, wireless handsets, tablets, and laptops, and Oracle Java ME Embedded represents the platform that will connect things to the internet. Depending on which analyst report you read, by 2020 there will be 30 to 60 billion things connected. Now, you can use your existing Java skills to program for these many devices.

Java Magazine: What is the Qualcomm Orion board?

Smith: The Orion board is Qualcomm's developer board for M2M and the Internet of Things. Since the inception of Qualcomm's new M2M business unit in early 2012, we have been working with Qualcomm on Oracle Java ME Embedded support for Orion.

Java Magazine: Why does this matter to a Java developer?

Smith: Unlike most developer boards, Orion is different in that the system on a chip [SoC] is attached to the board via a daughter card. Qualcomm has many different SoC combinations with different capabilities and modems that will provide developers one board that spans many different device types. Included on the motherboard are WiFi a/b/g/n, an SD card slot, four DB9 connectors, five LEDs, JTAG, USIM, GPIO, SPI, and I2C. The board even runs on the included battery. It also comes with sensors for temperature and light; GPS; and a 3-D accelerometer, so developers will be able to prototype solutions for container management, building automation, automotive, and the next wave of cool devices on the board. We also have a Java ME Software Development Kit integrated with the NetBeans IDE that emulates the features of the board. We really think that this is going to accelerate time to market for developers because they can get their hands on the platforms as soon as they are available, rather than waiting for OEM [original equipment manufacturer] prototypes. It's very exciting. Additional good news for developers: these prototype boards usually cost around $5,000 for a single solution, and with the Orion board they can get a scalable platform for around $500.

Kevin Smith of Oracle Technical Business Development explains what's new for partners and Java developers in the embedded space.

7 "GIVE US THE TOOLS, AND WE WILL FINISH THE JOB."*

Developing code requires tools; the better the tools, the easier the job is to finish and the more quickly it gets done. Today, time to market is just as critical as functionality for a product's success.

Tools for writing assembler and C code for embedded systems tend toward text editors and command-line tool chains controlled by esoteric tools such as Make. These work, but they severely limit the productivity of developers. Java developers have a great deal of choice of commercial as well as free and open source integrated development environments (IDEs). Tools such as NetBeans and Eclipse make writing code much easier and less error prone. Features such as automatic code completion, syntax checking as you type, and integrated documentation all help to get the job done more quickly and make the code more robust. As rich client platforms (RCPs), these tools are easy to extend, so they can be tailored to the needs of specific types of application development. Development time can be greatly reduced by building emulators for the target hardware to work with a desktop-based IDE.

* WINSTON CHURCHILL, BBC RADIO BROADCAST, FEBRUARY 9, 1941

8 DEVELOP ON ONE MACHINE, DEPLOY TO ANOTHER.

Embedded systems by their very nature tend not to resemble desktop or laptop systems. Frequently they are what is called *headless*, meaning that they have no display attached. This adds extra challenges and complexity to software development in terms of how to compile the code on one machine and deploy it to another.

Because Java uses a VM, it doesn't matter where you compile your code, so there's no need to set up complex cross-compilation tool chains. Use your favorite Java IDE to develop the code in comfort on your desktop, and then simply copy the class files or JAR files to the target device. To make developers' lives even easier, most IDEs support the concept of remote debugging through a network connection. If the code is doing what you told it to do, rather than what you wanted it to do, simply use the debugging facilities in your IDE to figure out what the issues are. This will be a lot faster than using print statements.

9 NINE MILLION DEVELOPERS CAN'T BE WRONG.

Depending on whose survey you consult, there are up to 9 million developers in the world who know how to program in Java. Almost all universities use Java as a teaching language for object-oriented programming fundamentals, which means that the number of Java developers is likely to increase.

If you want to develop Java code, you have a huge pool of programming talent to draw on. Because the JVM and the class libraries abstract away much of the complexity of developing embedded system code, you don't need to find people who have lots of experience in this field. For situations where more-domain-specific knowledge is required, there will still be a larger number of Java-capable developers to fill these positions.

BIG TALENT POOL

Because the JVM and the class libraries abstract away much of the complexity of developing embedded system code, you don't need to find people who have lots of experience.

10 FROM DEVICE TO DATA CENTER . . .

For embedded systems to have real value, they need to be networked, not just to other embedded systems but also to places where data can be aggregated, analyzed, and searched. This is what data centers are good at, recording and processing huge amounts of data from many different sources and then mining the data for useful information. What is the most popular platform for developing enterprise applications? Java. Using Java in embedded devices, in the data center for big data processing, and on the desktop for presenting and controlling the whole system makes architecting a complete solution simpler, quicker, and cheaper.

As you can see, Java offers benefits for developing code for embedded systems. With lower cost, more choices of platform, and readily available skilled developers, there's nothing holding you back from helping to build the Internet of Things. `</article>`

Simon Ritter is a Java technology evangelist at Oracle. He has worked in both Java technology development and consultancy. He now focuses on the core Java platform and Java for client applications.

(embedded)

JAVA ARRIVES ON A $25 BOARD

Eben Upton and the **Raspberry Pi Foundation** endeavor to inspire a new generation of programmers.

BY DAVID BAUM AND ED BAUM

You've heard the phrase before: "This thing is going to change the world." People said it about the electric lightbulb, the telephone, and the internal combustion engine. And they were right. But what does it take to change the world today? Can a single idea or innovation still do that? For Eben Upton, cofounder of the UK-based Raspberry Pi Foundation, the answers to those questions are linked to the way he hopes to measure the success of his project: by adding to the critical mass of students interested in computer science.

ART BY NICHOLAS PAVKOVIC, PHOTOGRAPHY BY JOHN BLYTHE

The Raspberry Pi fills the niche need for low-cost programmable computers, says Raspberry Pi Foundation Cofounder Eben Upton.

SNAPSHOT

RASPBERRY PI FOUNDATION
raspberrypi.org

Headquarters:
Caldecote, Cambridgeshire, England

Industry:
High technology

Employees:
3

Java technology used:
A special version of Java Virtual Machine adapted for the Raspberry Pi computer

The heart of the project is the Raspberry Pi, a tiny yet powerful general-purpose computer priced at US$25. The device is about the size of a deck of playing cards and can deliver 1,080p Blu-ray-quality graphics via its HDMI port. It runs Linux kernel–based operating systems.

Oracle is in the process of porting the Java Virtual Machine (JVM) to run on the Raspberry Pi's ARMv6 700 MHz processor, expanding the potential uses for the minicomputer and making it easier than ever for people to create lightweight, powerful, and reliable programs.

"Java is kind of the missing piece for us," Upton says. "Licensing agreements are underway that will enable us to bundle Java on the Raspberry Pi. When we have that, then we can claim that we have a full suite of mainstream development tools bundled on the device."

WHERE HAVE ALL THE CODERS GONE?

With processing power sufficient to drive a high-definition media center at the price of a couple of movie tickets, it's not surprising that the Raspberry Pi is on track to ship its millionth unit by the end of its first year in production. But Upton and his colleagues consider the number of units sold to be a crude measure of impact. For them, success is gauged on a philanthropic scale.

"How are we going to find tomorrow's computer programmers?" Upton asks, and with good reason. As director of studies in computer sciences at St. John's College, Cambridge, he has personally witnessed a steady decline in the number of applicants for computer sciences, not to mention a regression in the complexity of the programming tasks those applicants can undertake. He says the same trend can be seen at almost any computer company, where there are more engineers in their 30s than in their 20s and experienced programmers are often hired away from other companies because there aren't enough people entering the field out of college.

"One of the main reasons for the deterioration in computer skills in the upcoming generation is attributable to a simple fact: low-cost programmable computers that were available when I was young aren't

A CROSSOVER BOARD

This intriguing little device with its impulse-buy price tag is also gaining momentum among adult hobbyists, tinkerers, and developers. And it's raising eyebrows in industrial sectors, as managers begin to perceive its potential as a low-cost alternative to domain-specific devices.

around anymore," Upton explains. "We wanted to build a piece of hardware to fill that niche. The Raspberry Pi is cheap enough so that every student can have one. Some of them will be captivated by it. With Java on it, we'll be able to provide an environment that is friendly enough to enable a very young child to start programming with a language that professional software engineers use."

"Java lets students learn how to program without getting bogged down by the complexities of C++ and other languages," Upton adds. "Java tells you when you have done something wrong in your coding. And it is a professional programming tool. It's a nice addition to the Raspberry Pi."

SHAPING THE FUTURE ONE CHILD AT A TIME

The Raspberry Pi was developed primarily as an affordable education tool to inspire the next generation of computer programmers. Not surprisingly, the intriguing little device with its impulse-buy price tag is also gaining momentum among adult hobbyists, tinkerers, and developers. And it's raising eyebrows in industrial sectors, as managers begin to perceive its potential as a low-cost alternative to domain-specific devices.

But Upton keeps bringing the focus back to "units of one" and the long-term impact of getting these tantalizing devices into homes and schools where children are inspired to play with them, limited only by their imaginations. In that way he hopes to begin, through his foundation, to refill a pipeline of talent that he says society stopped filling in the late 1990s when programmable home computers such as the Commodore 64 and the BBC Micro were replaced by closed-platform gaming consoles and PCs and Macs with GUIs that don't expose users to a programming language or a command screen.

"We'll consider the project a success if we eventually see more kids applying for college in computer sciences," Upton says. "My hope is that we can add 1,000 computer science students per year in the UK. That would transform the industry and the lives of those students." Upton would like to see people begin to learn programming as early as age 8, apply to universities at 18, move into industry at 21, and start their own companies at 30. It's a long-range vision. Sometimes that's what it takes to change the world.

AN ENGAGING ENVIRONMENT FOR STUDENTS

The Raspberry Pi ships without a case, which naturally invites scrutiny and arouses curiosity. Unlike the slick, sealed units that dominate the world of personal computing, students can see the circuit board and components and learn to identify their various functions. The Raspberry Pi has a Broadcom BCM2835 system chip (which contains an ARMv6 700 MHz processor and a VideoCore IV GPU), 512 MB of RAM, an HDMI port, video and audio outputs, a USB port, composite video out, and an SD card slot for operating systems and applications. A US$35 model includes a second USB port and an Ethernet port for networking. The Raspberry Pi does not currently include wireless networking, but users are successfully connecting USB Wi-Fi adapters (dongles).

The major hurdle during the early days of the project was finding components that could deliver the requisite performance without boosting the price beyond the reach of many students. The breakthrough in delivering high-quality graphics came from

Get Started with Raspberry Pi

If you are ready to dive into programming on the Raspberry Pi, don't miss the article "Get Started with Java SE for Embedded Devices on Raspberry Pi," in the November/December 2012 issue of *Java Magazine*. Authors Bill Courington and Gary Collins provide an in-depth, step-by-step guide to getting Linux and Oracle Java SE Embedded running on your Raspberry Pi in less than an hour.

Broadcom Corporation, where Upton is employed as an SoC (system-on-a-chip) architect. In 2010, he was involved in the design and development of the Broadcom BCM2835 multimedia applications processor.

"Having decent graphics performance and multimedia features can attract children to the platform," Upton says. "Gaming and Blu-ray-quality movie playback are hooks to draw young users to the device."

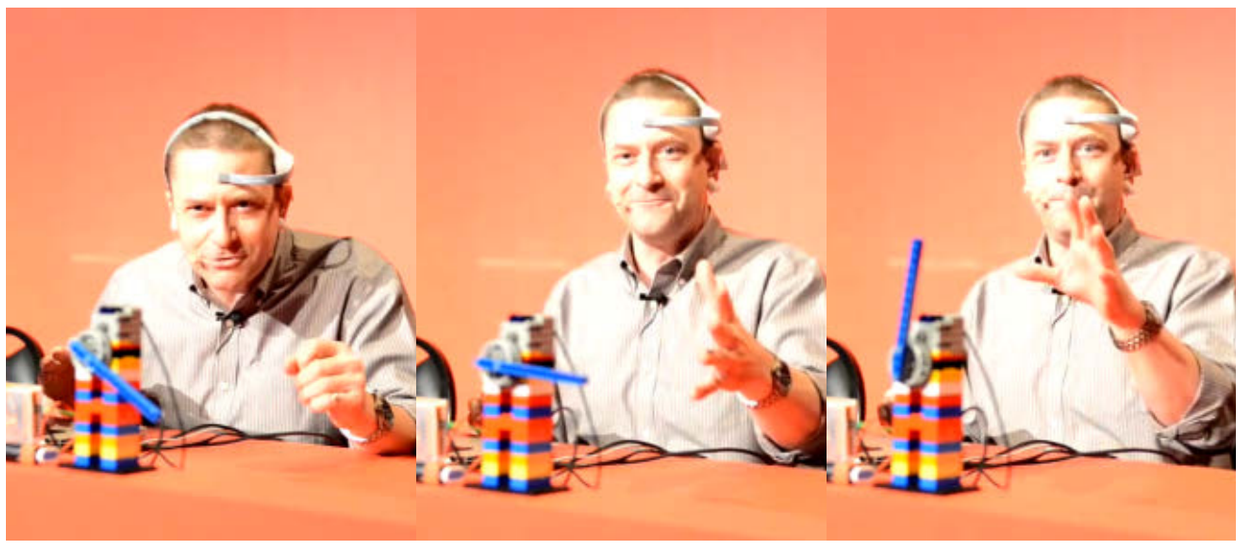
FINDING THE SWEET SPOT

While 512 MB of RAM may seem prohibitively small for a programming environment, Java actually fits well within those confines. "Java is a language that is ideal for creating small applications that run very portably," says Simon Ritter, technology evangelist at Oracle. "The Java development community is accustomed to these kinds of limitations. When Java was created back in the 1990s, most desktops had only 4 or 8 megabytes of memory."

Optimizing Java for the tiny computer's ARMv6 processor requires some finesse. The challenge is related to floating-point arithmetic—the way the processor performs arithmetic on real, or nondecimal, numbers. Many chips based on the ARMv6 architecture lack hardware floating-point units; the Broadcom BCM2835, however, does include ARM's virtual floating point (VFP) coprocessor.

A port of the JVM that uses soft floating point on ARM processors is currently available; it will work on the Raspberry Pi and is supported by Oracle. Oracle is creating a different version of the JVM that will run on the Raspberry Pi's processor and will take advantage of the acceleration for hard floating-point arithmetic. That should yield a 10 to 15 percent improvement in performance for more-compute-intensive applications such as those that use graphics. "We're in a comprehensive testing cycle for the optimized version of JVM for the Raspberry Pi," Ritter says. "That should deliver a lot of benefits for users."

What else does Java bring to this tiny computer? For Upton, it all comes back to educating children, the objective at the heart of the Raspberry Pi Foundation. "There are some high-quality educational applications that run on Java that we can bring across with no investment," he says. "That's very important to us."



Simon Ritter demonstrates mind control of a Lego windmill using only his brain (and a Raspberry Pi running Java).

Ritter concurs that the sweet spot for the Raspberry Pi is in education and that Java is well positioned to support that. "For students and hobbyists, Java provides better reliability and easier debugging than many of the other development languages that have been ported to the Raspberry Pi," he adds. "And there are many existing libraries of frameworks for Java that can be used in conjunction with the device. It has great potential to provide schools with a very low-cost platform that children can experiment with."

Of course, ultimately these students will move into the business world, and some of them will use their skills to adapt this low-cost computing platform for commercial purposes. Java's mature ecosystem will facilitate that process. "GlassFish Server Open Source Edition could be put onto the Raspberry Pi so it could serve Web pages or run Java enterprise applications," Ritter says, citing one example of a commercially viable Java companion. "GlassFish has been designed very carefully using a microkernel, so it does not require a lot of memory in its most basic form. It only uses the parts it needs to run the application. It is well suited to this kind of device."

Oracle is in the process of porting the JVM to run on the Raspberry Pi's ARMv6 700 MHz processor.

Upton also recognizes that his invention will soon expand beyond the classroom. "We're going to create a lot of value for people developing industrial applications," he says, pointing out that a small, affordable general-purpose computer could ultimately replace a wide range of specialized devices, especially those that require embedded microprocessors to perform essential tasks on the shop floor, on an assembly line, and for finely tuned scientific applications.

Yet wherever this useful device may end up, fostering creativity remains the driving force behind the Raspberry Pi Foundation. Java is expected to play an important role in that process, whether in schools, in industry, or in the hands of the do-it-yourself community.

LOOKING AHEAD

Future enhancements may include Wi-Fi plug-and-play with firmware revisions, an onboard Skype service, and add-on expansion boards for cameras.

"Once hardware is a given, people will be able to innovate at the software level," Upton concludes. "We are very hopeful that the Raspberry Pi and its educational initiatives will refill the pipeline of talented individuals who will shape the future for all of us." `</article>`

Based in Santa Barbara, California, **David Baum** and **Ed Baum** write about innovative businesses, emerging technologies, and compelling lifestyles.

(embedded)

THE FUTURE OF MONEY

With its MintChip offering, the **Royal Canadian Mint** plays the Java Card platform in the emerging world of digital currency.

BY PHILIP J. GILL

Minting coins is an activity as old as the earliest human civilizations. From the precious stones and metals that millennia ago replaced the barter of cattle, sheep, and grain, what serves as money has continued to evolve. Today the future of money appears to be the many forms of digital currency—money or scrip that is only exchanged electronically over private and public computer networks, including the internet.

For the Royal Canadian Mint (RCM), the sole legal producer of all circulation coins in Canada, the transition to digital currency represents a major opportunity. "We consider ourselves to be quite innovative at the Royal Canadian Mint," says Marc Brûlé, chief financial officer at the RCM, which is headquartered in Ottawa, the nation's capital.

Royal Canadian Mint CFO Marc Brûlé (left) and Dr. David Everett

ART BY NICHOLAS PAVKOVIC, PHOTOGRAPHY BY PATRICK FORDHAM

Brûlé and Everett surround themselves with coins at Royal Canadian Mint headquarters in Ottawa.

SNAPSHOT

ROYAL CANADIAN MINT
mint.ca

Headquarters:
Ottawa, Ontario, Canada

Industry:
Government

Employees:
More than 900 in 2011

Revenue:
CA$3.2 billion in 2011

Java technology used:
Java Card

"For years we have been following the advent of electronic payments and their growth," Brûlé continues. "Three or four years ago, we decided to dedicate some research and development funding to looking into electronic payments and, more specifically, what needs in the marketplace weren't being fulfilled and whether there was a role the Mint might play in this emerging market."

From that decision came a new digital currency, prototypes of which were introduced last year: the MintChip. Many forms of digital currency already exist—from electronic funds transfer (EFT) to direct deposit and loyalty cards—but the RCM sees the MintChip as a potential replacement for physical currency, both coins and paper, in Canada and other countries around the world. Being digital, says Brûlé, the MintChip can be denominated in almost any world currency.

"We see the MintChip as being an option to physical cash," says Brûlé. "We believe the MintChip can be very cost-effective in the low-value transactions market. It offers a good value proposition for merchants and consumers alike because it doesn't run across a network, is highly portable, and provides chip-to-chip secure communications."

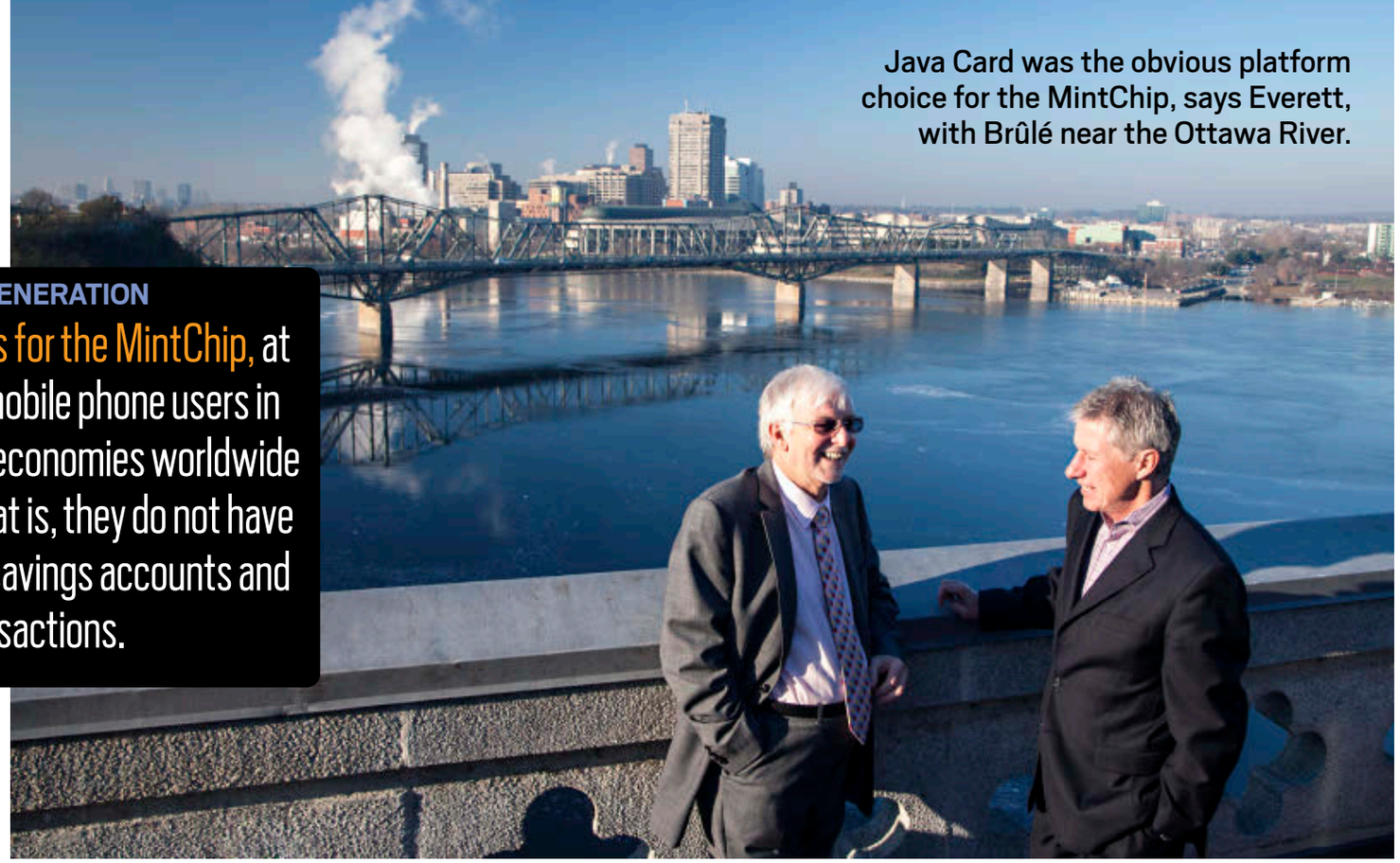
The MintChip is built entirely on Java Card technology, the de facto standard platform for smartcards and other small memory footprint devices. More than 12 billion Java Card–enabled devices have shipped since the platform was introduced in 1997, according to Brian Kowal, Java Card business manager at Oracle. "Java Card is the world's number one leading software application platform by volume," says Kowal. (See sidebar, "A Pocketful of Java Card.")

Kowal says the MintChip has one advantage that none of its rivals thus far have: "The MintChip is the only digital currency in the world being researched by a sovereign government," says Kowal. Given that the Canadian dollar is one of the world's most widely traded currencies, that adds substantial credibility.

POCKET CHANGE

The RCM was founded in 1908 as a branch of the UK Royal Mint and in 1969 was incorporated as a for-profit commercial crown corporation. In monarchical Commonwealth countries such as Canada, crown corporations are government-owned enterprises that undertake activities on behalf of their sole shareholder—in the RCM's case, the Canadian government. While the Bank of Canada, a separate crown corporation, issues all Canadian banknotes, the RCM mints the "pocket change" Canadians use every day to buy a cup of coffee, pay a transit fare, or purchase a newspaper or magazine. These include the "Loonie," a CA$1 coin so nicknamed because it bears the image of a common loon on one side, and the CA$2 "Toonie," whose nickname blends the words *two* and *loonie*.

Besides Canadian coins, the RCM also mints commemorative and collector coins, bullion coins, medals and medallions, and even coins for other sovereign governments—in any given year, the RCM contracts to manufacture coins for as many as a



Java Card was the obvious platform choice for the MintChip, says Everett, with Brûlé near the Ottawa River.

MONEY FOR THE MOBILE GENERATION

One of the target markets for the MintChip, at least initially, is younger mobile phone users in developed and emerging economies worldwide who are "unbanked"—that is, they do not have regular bank checking or savings accounts and regularly deal in cash transactions.

dozen foreign countries. "We operate like any other corporation," says Brûlé. "We operate to make a profit, and we receive no government funding."

In prototype form, the MintChip is an SD, microSD, UICC (Universal Integrated Circuit Card), or USB memory card containing an integrated circuit that provides security. It comes loaded with the MintChip application; the Java Card platform, including a Java Virtual Machine (JVM); and an operating system. Written in Java, the MintChip runs atop the JVM within the Java Card runtime, so there's nothing to stop it from being deployed in the cloud as well, adds Brûlé.

All Java Card platform chips—including those the MintChip uses—are highly secure, says Oracle's Kowal. They are certified under the US FIPS 140-2 and European Common Criteria EAL5+ standards, used for military and government IDs and payment systems worldwide. "What makes this interesting," says Kowal, "is that the MintChip could be embedded in an automobile, a smart meter, or a personal e-reader or tablet and provide payment solutions for all a consumer's personal connected devices."

"Java Card hardware has unique attributes to secure against physical attacks," continues Kowal. "They have small memory footprints, are low-power, and have a persistent memory model that holds transactions even if power is removed midway."

One of the target markets for the MintChip, at least initially, is younger mobile phone users in developed and emerging economies worldwide who are "unbanked"—that is, they do not have regular bank checking or savings accounts and regularly deal in cash transactions.

"The way the MintChip works is what we call an asset transfer model," explains Dr. David Everett, an independent consultant and the MintChip's chief technical architect. "The consumer actually holds the value in their chip. If you have a chip in your phone, for instance, the value's actually in that chip. And when you make a payment to somebody else, it leaves your chip and goes to another chip."

A typical use case for the MintChip would be mass transit, Everett says. Consumers load their MintChip

A Pocketful of Java Card

Odds are, if you have a smartcard in your pocket, it's a Java Card. More than 2 billion Java Card–enabled smartcards ship every year, and to date, more than 12 billion Java Cards have been issued since the technology platform was first introduced in 1998.

The success of the Java Card platform can be traced to several technical features, according to Tankar "Ravi" Ravishankar, principal engineer at Oracle. When Java Card technology was first released, Ravi explains, there were many competing platforms that were proprietary and which had long development cycles. "Java Card technology enabled developers to write applications in Java that could run on cards from different vendors, could be developed using off-the-shelf tools, and were less error prone," he says.

That portability also enabled Java Card developers to move applications from credit card–like smartcards to the new mobile phones that emerged in the early 2000s, he adds.

Because smartcards typically use persistent memory such as EEPROM or flash, Java Card technology includes persistent objects to meet their special needs, says Ravi. Smartcard chips also have no battery power; power is supplied to the chip when the card is placed in a reader device. "The Java Card platform features a transaction framework to ensure that the application state is always consistent," says Ravi.

The US Defense Manpower Data Center adopted Java Card technology for the US military and government departments. Ravi says this has given Java Card technology a high-security branding that has been instrumental to its growth and acceptance.

devices—credit card–like smartcards or mobile phones—with money from a kiosk, an ATM, or across the cloud from a PC or tablet. As they board a bus or train, they run their MintChip device through a reader of some kind, instantly transferring the fare amount from the chip in their card or mobile phone to the chip in the reader, with no intervening computer network or back-end bank approvals.

A NATURAL PLATFORM

Java Card technology is a natural platform for the RCM's digital currency venture. "For us, Java Card was a very obvious choice because the platform offered us quite a lot," says Everett. "There are other platforms, but the Java Card platform is ubiquitous, particularly in the mobile market. We've assumed that in general people will be using Java applications on Android and BlackBerry phones, iOS devices, PCs, and other similar mobile devices."

To spark interest among developers, he adds, "we have also provided APIs to the MintChip that allow developers to use their existing platform, such as Java on a mobile phone, which is very popular."

By necessity, MintChip devices will be highly secure, Everett adds. In addition to anticipated certification under the Common Criteria and FIPS 140-2, the MintChip assures users an extra layer of security not available in other electronic and digital currencies: because there's no network or approvals involved, no personal information is released between the devices and no personal information travels over a network. "MintChip transactions can be completely anonymous," says Everett.

A challenge for all smartcard platforms is performance. "In mass transit, you need to have a very fast performance time," says Everett. "This has always been a challenge with a virtual machine like Java Card to achieve that sort of 300-millisecond-type transaction, but we believe it will stand up."

Given these criteria, says Everett, "Java Card came out to be an outstanding winner."

To validate the MintChip technology and business propositions, the RCM announced the "MintChip Challenge," a competition for software developers to build digital currency applications using the MintChip. "We had decided we would reach out to 500 software developers," says Brûlé. "We weren't sure what the uptake would be, but it only took 72 hours to have all 500 spoken for."

In the end, 57 qualifying applications were submitted. The grand prize winner, announced in September 2012, was MintWallet, a cloud-hosted peer-to-peer network for sending and receiving virtual cash. Surprisingly, MintWallet is a Windows-based Java Card application, proving that the MintChip's digital currency is as portable as the real-world pocket change it could replace. `</article>`

Philip J. Gill is a San Diego, California–based freelance writer and editor who has been following Java technology for almost 20 years.

MAX BONBHEL

BIO

Part 3

Introduction to Web Service Security from Server to Client

Use Mutual Certificates Security to set up trusted authentication and protection that ensures the integrity and confidentiality of messages.

This article concludes a three-part series that focused on ensuring the security of Web services from server to client. We have explored how to secure Web services through the following mechanisms:

- In Part 1, we explored Username Authentication with Symmetric Key to generate a certificate/key couple shared by the client and the server to sign and encrypt messages. We also used the username/password combination to authenticate the client during service calls.
- In Part 2, we explored how to protect the data transiting between the client and the server using Secure Sockets Layer (SSL) via the secure HTTP transport, HTTPS.

Now we are going to use a mechanism that allows the client and the server to mutually authenticate.

Note: The complete source code for the application designed in this article can be downloaded here.

What Is the Mutual Certificates Security Mechanism?

Mutual certification, also called *shared certification*, is a process or technology where two communicating entities authenticate mutually. The Mutual Certificates Security (MCS) mechanism provides security through authentication and message protection to ensure integrity and confidentiality. This mechanism requires a keystore file and a truststore file for both the client and the server sides of an application.

The server certificate is provided to the client in advance of any communication, so both the client and the server can be sure they are dealing with legitimate entities, as shown in **Figure 1**.

Prerequisites

Download the following software, which was used to develop the application described in this article:

- NetBeans IDE 7.2 (available for download here)
- GlassFish 3.1.2.2 (available for download here)
- Metro 1.3 or higher (included in NetBeans)

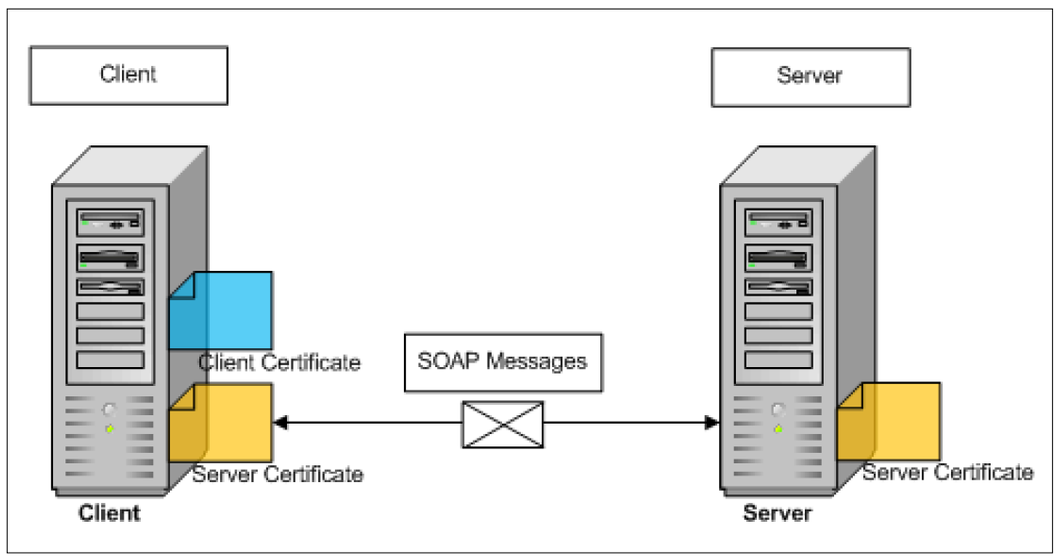


Figure 1

PHOTOGRAPH BY
ALLEN MCINNIS/GETTY IMAGES

Note: This article was tested using the latest version of the NetBeans IDE (version 7.2, at the time this article was written).

Overview of Adding MCS to the Web Service

What we are going to do is secure both sides of the AuctionApp application by using the MCS mechanism. Specifically, we will configure the keystore and the truststore for the client and the server to ensure the integrity and confidentiality of the message.

We will perform the following tasks:

- Secure the Web service by adding the MCS mechanism and configuring the keystore and the truststore for the server
- Create and configure a new Web client that references the Web service by specifying the secure Web Services Description Language (WSDL) file
- Test the noncertified client application
- Configure the keystore and the truststore for the client
- Test the certified client application

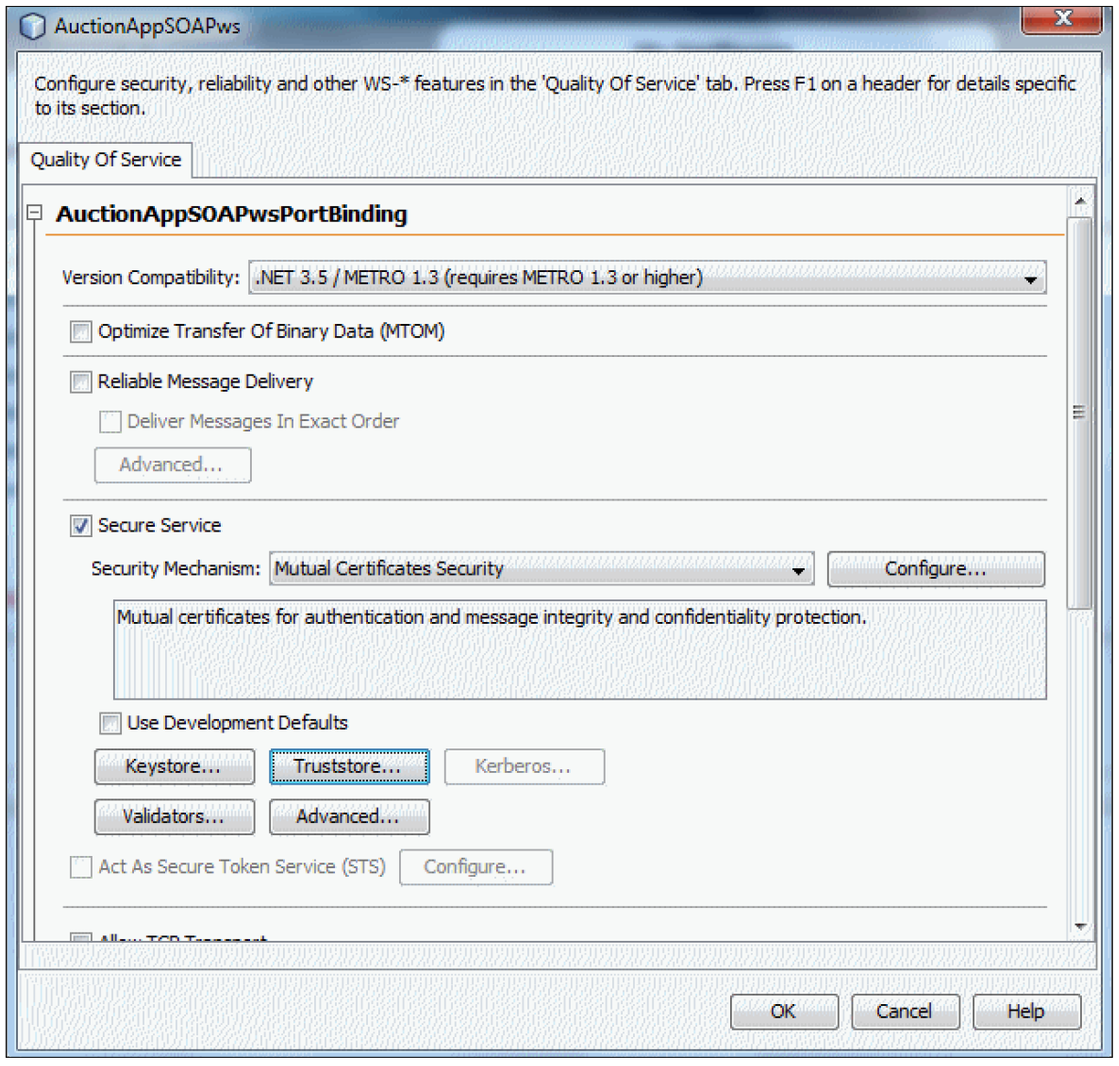


Figure 2

Note: The application we are going to secure is an online auction place (like eBay) that we created in a previous series of articles ("Introduction to RESTful Web Services"). Sellers post their items in listings, and buyers bid on the items. A seller can post one or many items, and a buyer can bid on one or many items.

Specifically, we are going to limit access to the Java API for XML Web Services (JAX-WS) Web service that extrapolates the amount of a bid by multiplying by a factor (100).

Add MCS to the Web Service

It will take less than five minutes to add MCS to the Web service using NetBeans IDE 7.2.

1. Add the MCS security mechanism to the AuctionApp application:
   a. Open the AuctionApp project in NetBeans IDE 7.2 or higher.
   b. Expand the **Web Service** node of the AuctionApp project and right-click the **AuctionAppSOAPws** node; then select **Edit Web Service Attributes**.
   c. Under the Quality Of Service tab, expand the **AuctionApp SOAPwsPortBinding** section.
   d. Make sure the **Reliable Messaging Delivery** option is *deselected*.
   e. Select **Secure Service** and then select **Mutual Certificates Security** from the Security Mechanism list, as shown in **Figure 2**.
2. Configure the keystore to specify the alias identifying the server certificate and private key for the AuctionApp application:
   a. Deselect **Use Development Defaults** if it is selected.
   b. Click the **Keystore** button, and then click the **Load Aliases** button and select **xws-security-server**, as shown in **Figure 3**.
   c. Click **OK**.

   At this point, NetBeans creates a new Web Services Interoperability Technologies (WSIT) configuration that contains the security elements within the sc:KeyStore tags of the wsit-com.bonbhel.oracle.auctionApp.AuctionAppSOAPws.xml file, as shown in **Figure 4**. The file is located in the Web

Pages/WEB-INF node of the AuctionApp project.

3. Deploy the service and display the WSDL file so you can inspect the sp:AsymmetricBinding tags:
   a. Right-click the **AuctionApp** node and choose **Deploy**.
   b. Open your browser and type the WSDL URL: http://localhost:8080/AuctionApp/AuctionAppSOAPws?wsdl. The Web service presents its WSDL file, as shown in **Figure 5**.

The sp:AsymmetricBinding tags contain all we need to encrypt and sign the SOAP message using asymmetric key algorithms (public/private key combinations). This tag should contain nested elements such as the following:

- <sp:InitiatorToken>: The *initiator token* refers to the public/private key-pair owned by the initiator.
- <sp:X509Token>: This tag specifies that the X.509 certificate token profile will be used. It contains the sp:IncludeToken attribute, which is responsible for adding the public key to the message sent to the recipient.

Create a New Web Client

In this section, we are going to create and secure a new Web service client that references the Web service that we just secured.

To do this, we will create a sample Web client application by using the Web Service Client wizard provided by NetBeans IDE 7.2 to generate the code and everything we need for looking up a Web service.

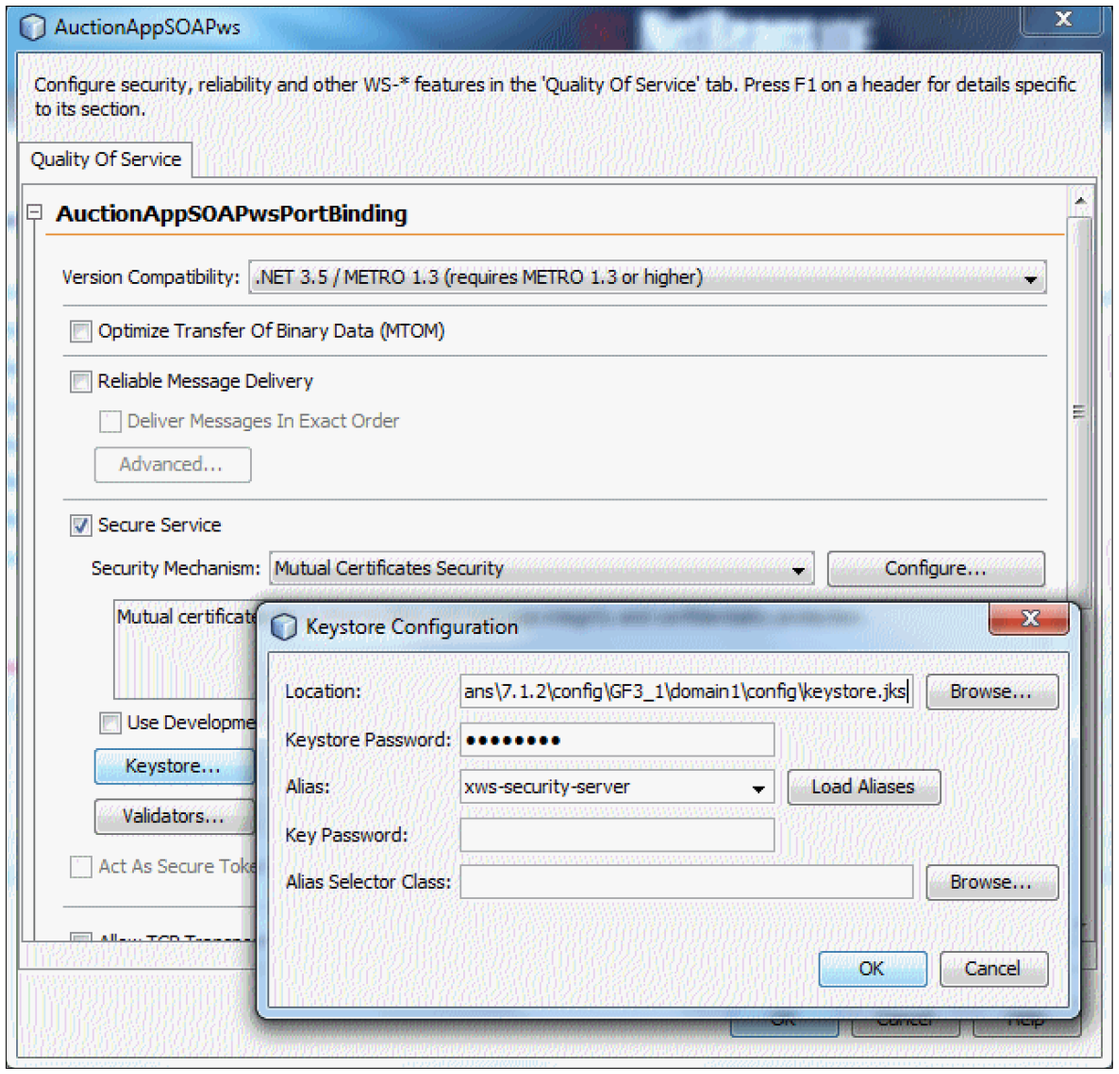


Figure 3

```
<sc:KeyStore wspp:visibility="private"
             alias="xws-security-server"
             storepass="changeit" type="JKS"
             location="...\.netbeans\7.1.2\config\GF3_1\domain1\config\keystore.jks"/>
```

Figure 4

```
▼<sp:AsymmetricBinding>
  ▼<wsp:Policy>
    ▶<sp:AlgorithmSuite>...</sp:AlgorithmSuite>
     <sp:IncludeTimestamp/>
    ▼<sp:InitiatorToken>
      ▼<wsp:Policy>
        ▼<sp:X509Token sp:IncludeToken="http://docs.oasis-open.org/ws-sx/ws-
         securitypolicy/200702/IncludeToken/AlwaysToRecipient">
          ▼<wsp:Policy>
             <sp:WssX509V3Token10/>
           </wsp:Policy>
         </sp:X509Token>
       </wsp:Policy>
     </sp:InitiatorToken>
    ▶<sp:Layout>...</sp:Layout>
     <sp:OnlySignEntireHeadersAndBody/>
    ▼<sp:RecipientToken>
      ▼<wsp:Policy>
        ▼<sp:X509Token sp:IncludeToken="http://docs.oasis-open.org/ws-sx/ws-
         securitypolicy/200702/IncludeToken/Never">
          ▼<wsp:Policy>
             <sp:RequireIssuerSerialReference/>
             <sp:WssX509V3Token10/>
           </wsp:Policy>
         </sp:X509Token>
       </wsp:Policy>
     </sp:RecipientToken>
  </wsp:Policy>
</sp:AsymmetricBinding>
```

Figure 5

Let's code this application in five minutes.

1. Generate the initial NetBeans project:
   a. From the File menu, choose **New Project**.
   b. From Categories, select **Java Web**.
   c. From Projects, select **Web Application**.
   d. Click **Next**.
   e. Type AuctionAppWebServiceClient for the project name and click **Next**.
   f. Make sure the server is specified as GlassFish Server (or similar wording).
   g. Click **Finish**.
2. Create the Seller entity:
   a. Right-click the **AuctionAppWebServiceClient** project and select **New**; then select **Entity class**.
   b. Type Seller in the **Class Name** field, type com.bonbhel.oracle.AuctionAppWebServiceClient in the **Package** field, and click **Next**.
   c. In the Provider and Database screen, select **EclipseLink (JPA 2.0)(default)** from the Persistence Provider list.
   d. From the Data Source list, select **jdbc/sample**, which is the datasource provided by NetBeans.
   e. Click **Finish**.
3. Perform actions similar to Step 2 to create the Item and Bid entities.
   Now we are going to add properties to the entities using the NetBeans wizard.
4. Open the Seller.java file, right-click anywhere in the code, and select **Insert code**.
5. From the Generate wizard, select **Add property**, and add the seller properties (String lastName, String firstName, and String email).
6. Open the Item.java file and add the item properties (String title, String description, Double initialPrice, and Seller seller).
7. To define the entity relationship, click the NetBeans warning (lightbulb) and select **Create bidirectional ManyToOne relationship**.
   This action creates a list of items in the Seller entity.
8. Open the Bid.java file and add the item properties (String bidderName, Double amount, and Item item).
9. To define the entity relationship, click the NetBeans warning and select **Create bidirectional ManyToOne relationship**.
10. Generate the Getters and Setter, respectively, for the list of items and bids created in the Seller and item entities by right-clicking anywhere in the code and selecting **Insert code**.
11. From the Generate wizard, select **Getter and Setter**.
    At this point, your Seller.java file will look like **Listing 1**.
12. The client implementation we are going to build consists of JavaServer Faces (JSF) pages based on the entities just created. Create the JSF pages:
    a. Right-click the **AuctionAppWebServiceClient** project and select **New**; then select **JSF Pages from Entity Classes**, click **Add all**, and click **Next**.
    b. For **Session Bean Package**, type a name such as com.bonbhel.oracle.AuctionAppWebServiceClient.facade, and for **JSF Classes Package**, type a name such as com.bonbhel.oracle.AuctionAppWebServiceClient.controller.
    c. For **Folder Name**, type a name such as jsfClient.

LISTING 1

```
@Entity
public class Seller implements Serializable {
  @OneToMany(mappedBy = "seller")
  private List<Item> items;
  private static final long serialVersionUID = 1L;
  @Id
  @GeneratedValue(strategy = GenerationType.AUTO)
  private Long id;
  protected String firstName;
  protected String lastName;
  protected String email;

  public List<Item> getItems() {
    return items;
  }

  public void setItems(List<Item> items) {
    this.items = items;
  }
```

Download all listings in this issue as text

d. Click **Finish**.

13. We are going to use the Web Service Client wizard to create the Web service client. We will assume that the JAX-WS Web service is an external service that resides in the application tier over the network. So, we will use the URL to the JAX-WS Web service WSDL file. Create the Web service client:

a. Make sure the AuctionApp project is up and running. If it is not, right-click the **AuctionApp** node and choose **Deploy**.

b. Right-click the **WebServiceClientSecureSSL** node and choose **New**; then select **Web Service Client**.

c. From the New Web Service Client wizard, select **WSDL URL**, and then specify the URL to the Web service WSDL file using the fully qualified host name, for example, http://localhost:8080/AuctionApp/AuctionAppSOAPws?wsdl, as shown in **Figure 6**.

d. Accept all other default settings. The package name will be taken from the WSDL file.

e. Click **Finish**.

14. Add the extrapolateAmountBid operation provided by the JAX-WS Web service to the client classes:

a. Open the **Source Packages** node of the AuctionAppWebServiceClient project and double-click the **BidController.java** file located in the controller package com.bonbhel.oracle.AuctionAppWebServiceClient.controller.

b. Put your cursor anywhere inside the source editor.

c. Expand the **Web Service References** node of the AuctionAppWebServiceClient project and drag the **extrapolateAmountBid** node inside the source editor. The extrapolateAmountBid() method appears at the end of the BidController class code, as shown in **Listing 2**.
Note: Alternatively, you can right-click anywhere in the source editor and choose **Insert Code**; then select **Call Web Service Operation** and click the **extrapolateAmountBid** operation in the Select Operation to Invoke dialog box.

15. We need to add some code application logic in the BidController class in order to extrapolate the amount of the bid (by factor, which is 100) when the user edits or views the bid entry. So call the extrapolateAmountBid operation to extrapolate the bid:

a. Open the BidController.java file in the source editor.

b. Modify the public String prepareView() method, as shown in **Listing 3**.

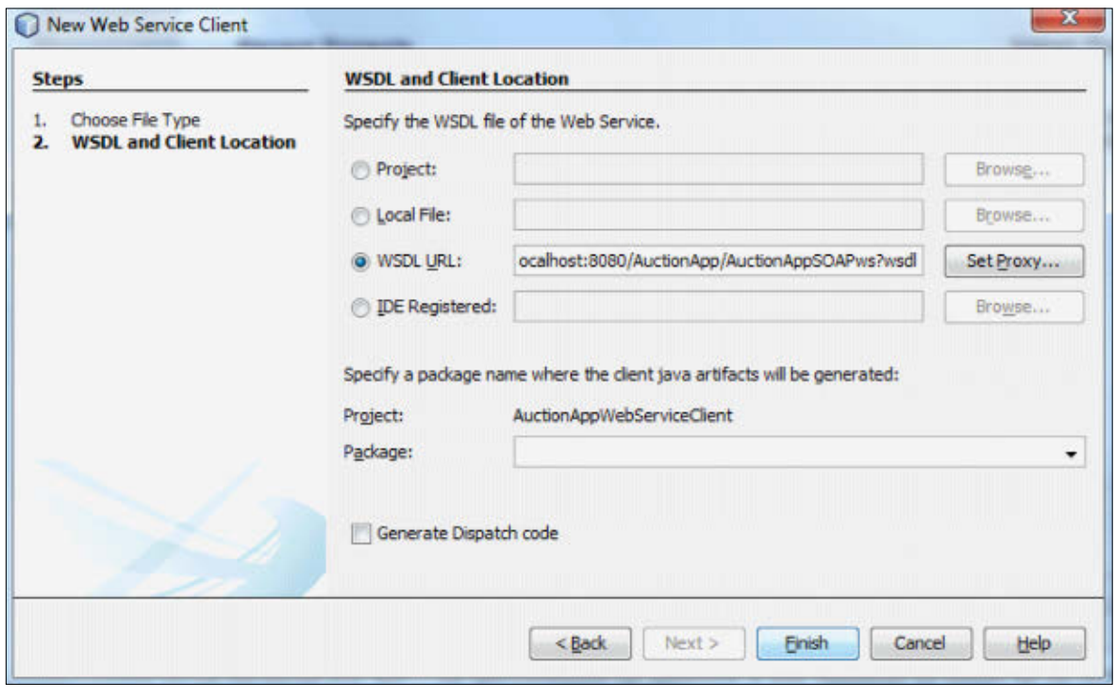


Figure 6

LISTING 2 LISTING 3

```
private Double extrapolateAmountBid
(double amountBid, int factor) {
com.bonbhel.oracle.auctionapp.ws.AuctionAppSOAPws
port = service.getAuctionAppSOAPwsPort();
return port.extrapolateAmountBid(amountBid, factor);
}
```

Download all listings in this issue as text

At this point, the secured Web service client is regenerated and references the secured Web service WSDL file.

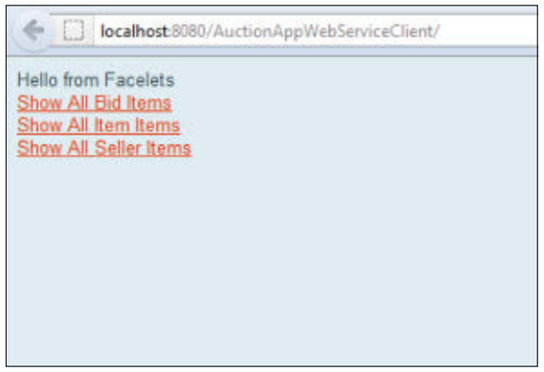


Figure 7

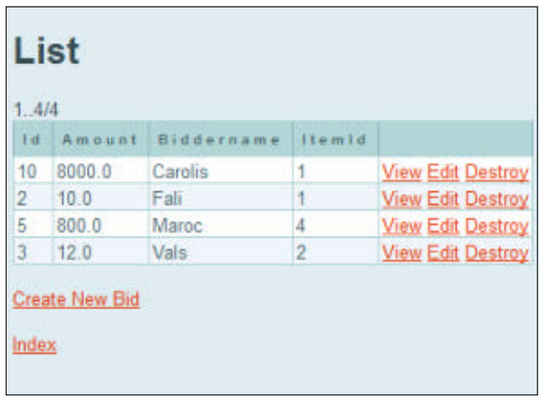


Figure 8

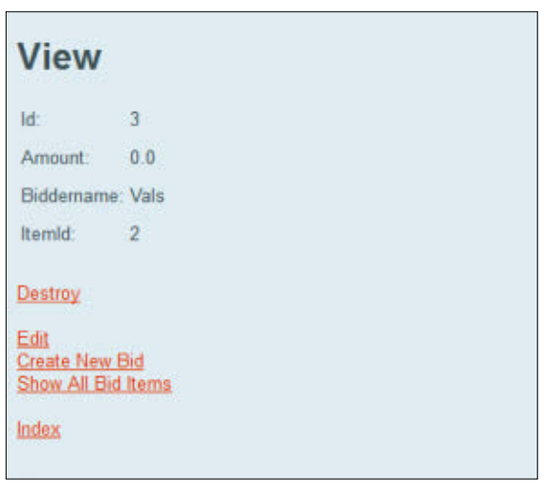


Figure 9

Test the Noncertified Client Application

Now it's time to test the service. We will invoke the secured Web service from the client application before we provide the server certificate to the client.

Try to invoke the secured Web service from the noncertified client to display the extrapolated amount of a bid:

1. Make sure the AuctionApp project is up and running. If it is not, right-click the **AuctionApp** node and choose **Deploy**.
2. Open the **Web Service References** node of the AuctionAppWebServiceClient project and right-click the **AuctionAppSOAPws** node; then select **Refresh**.
3. From the Confirm Client Refresh wizard, make sure **Also replace local wsdl file with original wsdl located at:** is selected; then click **Yes**.
4. Right-click the **AuctionAppWebServiceClient** project and choose **Clean and Build**.
5. Right-click the **AuctionAppWebServiceClient** project again and choose **Run**. The list of all entries is displayed, as shown in **Figure 7**.
6. Click the **Show All Bid Items** link to display the list of bid entries, as shown in **Figure 8**.
7. Click the **View** link for the bidder named Vals to see the newly extrapolated amount of the Vals bid, as shown in **Figure 9.**

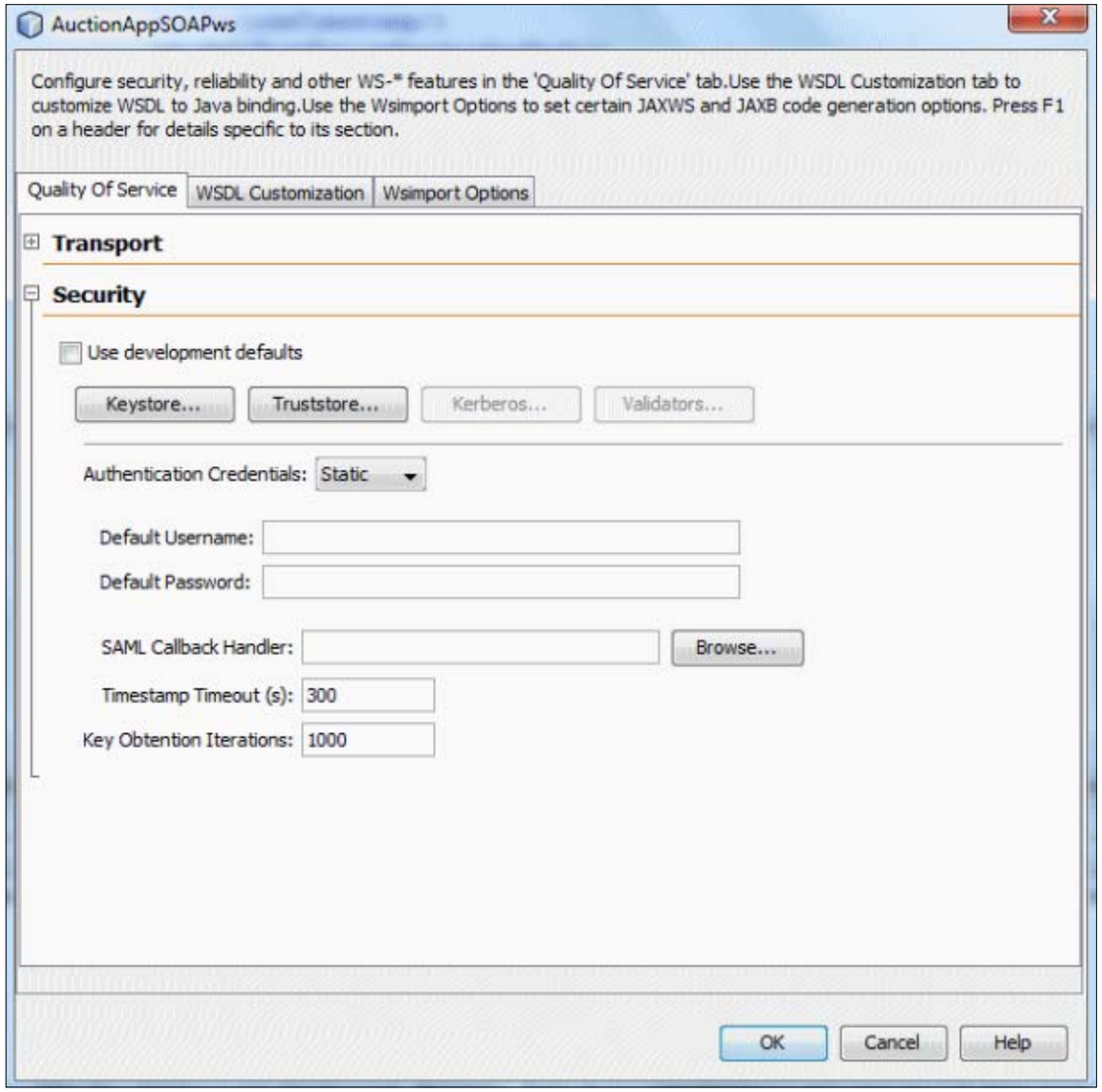


Figure 10

As you can see, the amount of the bid changed from 12.0 to 0.0. This means that the client failed to call the secured Web service.

Configure the Keystore and Truststore for the Client

In this section, we are going to provide the server certificate to the client by performing the following tasks:

- Configure the keystore to point to the alias for the client certificate
- Configure the truststore that

contains the certificate and trusted roots of the server

1. Configure the keystore for the client:
   a. Make sure the AuctionApp project we opened previously is up and running. If not, right-click the **AuctionApp** node and choose **Deploy**.
   b. Expand the **Web Service References** node of the AuctionAppWebServiceClient project and right-click the **AuctionAppSOAPws** node; then select **Edit Web Service Attributes**.
   c. Under the Quality Of Service tab, select the **Security** section, as shown in **Figure 10**.
   d. Deselect the **Use development defaults** option if it is selected.
   e. Click the **Keystore...** button, click the **Load Aliases** button, and select **xws-security-client**, as shown in **Figure 11**.
   f. Accept all other default settings and click **OK**.

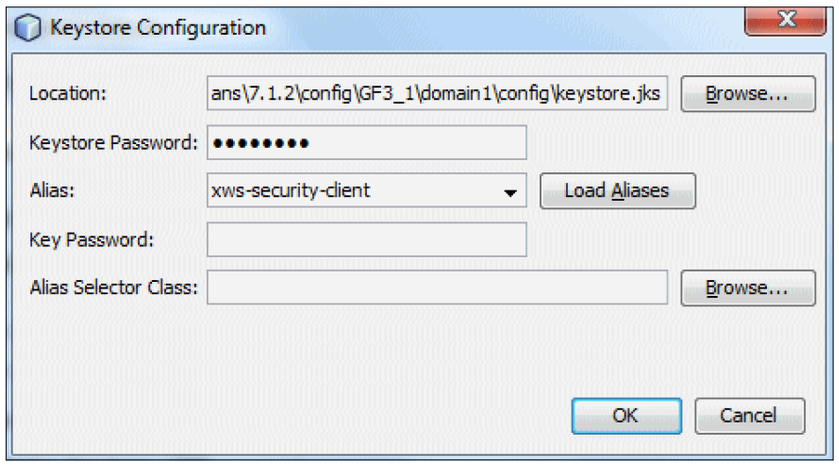


Figure 11

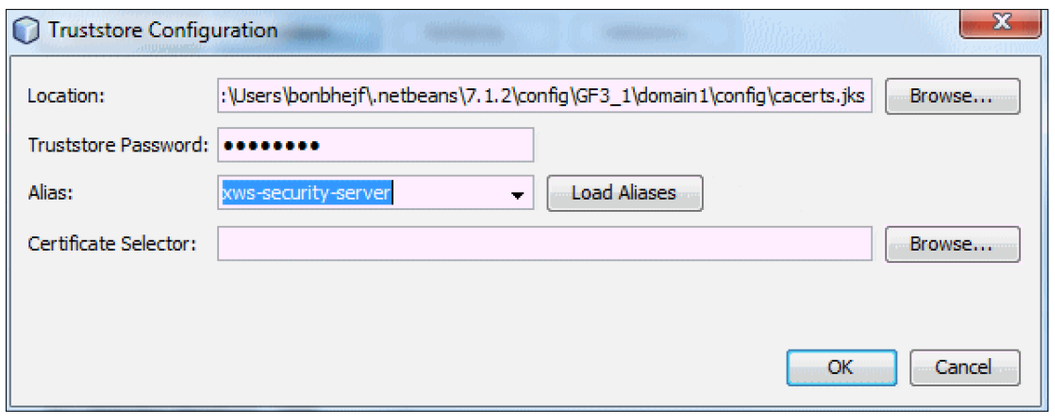


Figure 12

2. Configure the truststore for the client:
   a. Click the **Truststore...** button, click the **Load Aliases** button, and select **xws-security-server**, as shown in **Figure 12**.
   b. Accept all other default settings and click **OK**.
   c. Compile the application by right-clicking the **AuctionAppWebServiceClient** node and selecting **Build**.

At this point, the client and the server are authenticated with certificates. The client keystore is pointing to the alias for the client certificate, and the truststore contains the certificate and trusted roots of the server.

Test the Certified Client Application

Now we are ready for the real test. We are going to try to invoke the secured Web service from the certified client to display the extrapolated amount of a bid using the MCS mechanism:

1. Make sure the AuctionApp project is up and running. If it is not, right-click the **AuctionApp** node and choose **Deploy**.
2. Right-click the **AuctionAppWebServiceClient** project and choose **Deploy**.
3. Right-click the **AuctionAppWebServiceClient** project again and choose **Run**.
   The list of all entries is displayed, as shown in **Figure 13**.
4. Click the **Show All Bid Items** link to display the list of bid entries, as shown in **Figure 14**.
5. Click the **View** link for the bidder named Vals to see the newly extrapolated amount of the Vals bid, as shown in **Figure 15**.

As you can see, the amount of the bid changed from 12.0 to 1200.0. This means that the client succeeded in calling the secured Web service. Bravo!

SECURITY TOOL

The Mutual Certificates Security mechanism ensures integrity and confidentiality by requiring a keystore file and a truststore file for both the client and the server sides of an application.

Conclusion

In this article, we have seen how easy it is to add a new security layer to an existing application to protect the communication between the client and the server.

This series of articles provided an overview

of the principal aspects of Web services through the following mechanisms:

- Using a username/password combination to access the service
- Protecting the data transiting between the client and the server
- Establishing a mutual certificate between the client and the server

As demonstrated, NetBeans and GlassFish make adding security for Web services easier than ever. `</article>`

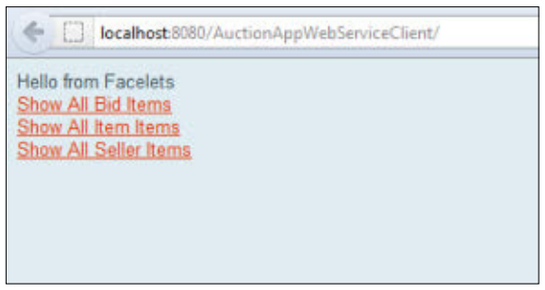


Figure 13

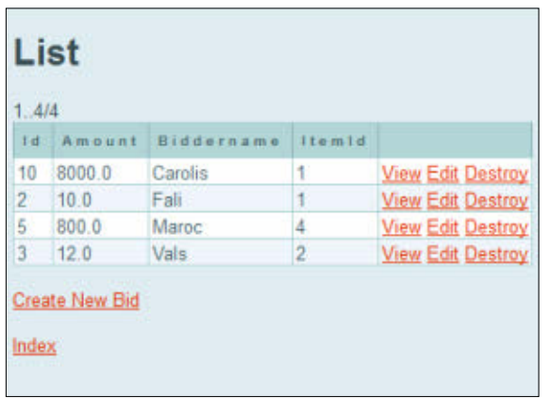


Figure 14

Figure 15

LEARN MORE

- NetBeans *Advanced Web Service Interoperability* manual
- *Metro User Guide*
- GlassFish resources

Oracle Press™

Your Destination for Java Expertise

Written by leading technology professionals, Oracle Press books offer the most definitive, complete, and up-to-date coverage of Oracle products and technologies—including the latest Java release.

Acclaimed programming author Herb Schildt's books have sold more than 3.5 million copies worldwide

Java: The Complete Reference, Eighth Edition

Herb Schildt

A fully updated edition of the definitive guide for Java programmers

Java: A Beginner's Guide, Fifth Edition

Herb Schildt

Essential Java programming skills made easy

Java Programming

Poornachandra Sarang

Learn advanced skills from an internationally renowned Java expert

Available in print and e-book formats

Join the Oracle Press Community at
www.OraclePressBooks.com

twitter @OraclePress

Part 1

Demystifying invokedynamic

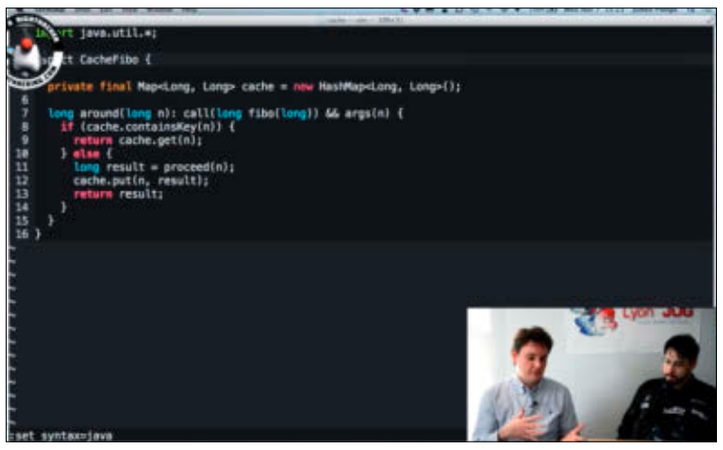
Learn how to use invokedynamic in your code.

JULIEN PONGE

BIO

The release of Java 7 brought substantial additions to the language, such as the new try-with-resources statement, multi-catch clauses, and the diamond operator. New APIs have been introduced, too, with notable examples being the fork-join framework for parallel algorithms and NIO.2 to better deal with native file system capabilities.

Another much-touted change is the introduction of a new Java Virtual Machine (JVM) bytecode instruction called invokedynamic. It was introduced to facilitate the implementation of dynamic languages on top of the JVM, which is an attractive target platform with a robust adaptive runtime that, in recent years, has seen many languages flourish. Some are ports of existing languages to the JVM (for example, Rhino, Jython, and JRuby) while some are new (for example, Groovy, Clojure, and Fantom).



Julien Ponge chats with Oracle's Stephen Chin about invokedynamic as part of Chin's NightHacking Tour across Europe.

PHOTOGRAPH BY
MATT BOSTOCK/GETTY IMAGES

This article is a gentle introduction to using invoke dynamic in your own code. The topic in itself is fairly rich and will mostly appeal to language and middleware implementers. We will only scratch the surface of the provided APIs and possible usages, but by the end you should have a basic technical understanding of invokedynamic.

Note: The source code for the examples in this article can be downloaded here.

Invoking Methods on the JVM

The JVM has traditionally offered four instructions for invoking methods. Each points out an owner class, a name, and a signature description. As an example, given a static method void baz(int a, String s) in class foo.Bar, invoking the method requires a reference with foo/Bar as the owner, baz as the name, and (ILjava/lang/String;)V as the JVM internal representation signature. Invoking a method requires placing a *receiver* object—that is, the object on which the method shall be invoked—on the stack. The only exception is the case of static methods, which do not have a receiver.

The following are the four invoke instructions:

- invokestatic, which is used for static methods.
- invokevirtual, which is used for methods that require dynamic dispatch, that is, public and protected methods.
- invokeinterface, which is similar to invokevirtual, except that the method dispatch is based on an interface type.
- invokespecial, which is for the other types of methods: constructors and private virtual methods. The method dispatch is

not performed based on the receiver type but rather based on the specified owner class (this is how the super reference works).

The design of the JVM makes perfect sense because it was originally created for a strongly typed language, Java. In such a language, it is sufficient to check method invocations at compile time. Hence, it is both easy and natural to produce bytecode where invocations unambiguously specify a target. Signature descriptors exactly match those of the target methods.

The Case of Dynamic Languages

Things are slightly different with loosely typed dynamic languages such as Groovy, JRuby, or Jython. In such languages, types are most often not checked at compile time. It is up to the programmer to pass a "good" type, and failing to do so results in an error at runtime. Thus, in such languages, the closest "accurate" JVM type for a method argument is more often than not java.lang.Object. It easily follows that this complicates type analysis (which has to be done mostly at runtime) and affects performance.

Dynamic languages also often support a form of *monkey-patching*, that is, the ability to add methods and fields to and remove methods and fields from existing classes. Some languages even support doing so at the instance level. For all these reasons, it is necessary to defer the resolution of classes, methods, and fields to the runtime. It is also necessary to dynamically update the resolved entities at runtime. Also, there is often a need for adapting a method invocation to a target of a different signature.

Dynamic languages on the JVM have traditionally built ad hoc runtime support on top of the JVM. This includes using wrapper type classes; building custom *inline caches* for invoking methods; interpreting, tracing, and generating specialized bytecode; or using hash tables to provide dynamic symbol resolution. In doing so, such languages generate bytecode with entry points to the runtime in the form of method calls using either of the four existing bytecode instructions for invocations.

Here Comes invokedynamic

The main problem is that of performance. The bytecode generated by dynamic languages tends to require several actual JVM method invocations in order to perform one in the dynamic language. There is extensive use of reflection and dynamic proxies, which have a performance cost. Last but not least, such code tends to target common language runtime classes and methods.

In short, because there are many different execution paths with great variance in actual types, the adaptive runtime—just-in-time (JIT) compilation—has trouble applying optimizations. Although dynamic language implementers have used clever techniques, it remains true that the JVM has issues understanding what such code does, thus hindering performance. The bytecode emitted by such languages lacks information and patterns that the JVM knows how to optimize, which are typically present in bytecode generated from Java. One can argue that this is a matter of making the JVM more aware of new patterns, but the fact that bytecode is typed also gets in the way.

The introduction of invoke dynamic to the JVM specification is recognition of the fact that dynamic languages bring value to the Java ecosystem and, as such, the JVM needs to provide better support for them. This new bytecode instruction is able to defer the invocation target resolution to some runtime logic, and it provides support for dynamically changing call site targets. It also has support in the JVM internals so that it can be better optimized by the JIT compiler.

Method Handles

Method handles are analogous to the function pointers found in languages such as C in the sense that they point to a target and can be used to invoke it. Method handles can be obtained and manipulated by using the new java.lang.invoke API.

Getting and invoking. Listing 1 is an example in which we obtain a method handle to the Boolean startsWith(String) instance method of the java.lang.String class and then invoke it.

We first need an object of type MethodHandles.Lookup to search for targets that include virtual methods, static methods, special methods, constructors, or field accessors.

A lookup object is dependent on the invocation context of a call site. As an example, a lookup made in a class A cannot see the private methods of a class B. When invoking the lookup method in **Listing 1**, we obtain a lookup object that respects the accessibility rules from within the main method of Sample1. The preferred way to look up only public methods is to call MethodHandles.publicLookup.

Once this is done, we can call the findVirtual method. The first argument is the class in which to look for the method. The second argument is the method name,

while the third one is an instance of MethodType. Here, the method that we are looking for returns a Boolean and takes a string as a parameter.

Note that we actually pass *three* arguments to build the type: the return type followed by the argument types. Indeed, startsWith is a virtual method, so its first call argument must be the *receiver* type, which here is a string, too.

Finally, we perform two invocations by printing whether *Java* and *Groovy* start with *J*.

java.lang.reflect integration. Just looking at the example in **Listing 1**, you could think that method handles can be used in place of reflective method invocations. Worse, you could think that java.lang.invoke is a possible replacement for reflection. This is far from the truth, because both APIs complement each other.

You need to know the exact signatures of the elements to be looked up using MethodHandles.Lookup. There is no way to look for the methods, constructors, and fields of a class just by their names without knowing their precise signatures. To do that, you should simply stick to the well-known reflective APIs in java.lang.reflect, and then use either of the unreflect methods found in MethodHandles.Lookup that provide bridging between reflection and method handles. This is especially useful when writing dynamic applications where you need to look up elements by name, but the exact signatures are not known.

- The example in **Listing 1** with method handles can be equivalently coded as shown in **Listing 2**, in which the target method handle is found using reflection.

If we were to stop our explorations of invokedynamic here, there would not be many differences between an invocation using reflection and one using a method handle. Still, it should be noted that every reflective invocation induces verifications at the JVM level. This is not the case with method handles, because those verifications happen only when method handles are built.

Combinators. While method handles can directly map to class methods, it is possible to build more-elaborate method handles by combining several of them. This is useful in many situations, such as when there is a need to adapt a call site and a target of different types, to process arguments and return values, or to provide conditional branching. In this regard, guards are an especially useful building block for constructing more-complex call-site logic as is found in inline caches.

LISTING 1 LISTING 2 / LISTING 3

```
package sample1;

import java.lang.invoke.MethodHandle;
import java.lang.invoke.MethodHandles;
import java.lang.invoke.MethodHandles.Lookup;
import static java.lang.invoke.MethodType.methodType;

public class Sample1 {
 public static void main(String... args) throws Throwable {
  Lookup lookup = MethodHandles.lookup();

  MethodHandle startsWith = lookup.findVirtual(
   String.class, "startsWith",
       methodType(boolean.class, String.class));
  // Prints "true"
  System.out.println(startsWith.invokeWithArguments(
   "Java", "J"));
  // Prints "false"
  System.out.println(startsWith.invokeWithArguments(
   "Groovy", "J"));
 }}
```

Download all listings in this issue as text

Let's start with a simple example. In **Listing 2**, we obtained a method handle to the startsWith method of the String class. Now, in **Listing 3**, let's instead build a method handle that checks whether a string given as a receiver starts with *J*.

To do this, we need to perform a form of *partial function application* that binds the first invocation argument of the method to the constant value *J*. The java.lang.invoke.MethodHandles class provides many *combinators*, that is, methods to derive new method handles from existing ones. In our case, we take advantage of insertArguments to bind the element at index 1. This method works as follows: it takes a method handle, an index where arguments will be inserted, and a variable arguments list of values to insert.

The other combinators can remove elements; permute ele-

ments; wrap arguments to or unwrap arguments from an array; perform type conversions; and more. While it is beyond the scope of this article to provide a complete overview of each of these, you will discover some commonly used ones if you keep reading.

Example of call site and target adaptation. You will often run into the problem of adapting call sites and targets when working with invokedynamic. Suppose that you have a method printAll declared as shown in **Listing 4**.

Clearly, printAll is of type (String, Object[])void. In Java, invocations of this method would call sites such as those shown in **Listing 5**.

Now suppose that we have call sites expecting something different, such as a method that takes a fixed number of parameters and no prefix string. Clearly, the signatures do not match. A fix would be to introduce an adaptation method such as that shown in **Listing 6**.

Such an adaptation is simple to perform using method handles and combinators, as shown in **Listing 7**.

First, target is a direct method handle to the printAll static method. We use the bindTo combinator to bind the first argument to a value. In the case of a static method, this is effectively the first argument, while in the case of an instance method, the object would be the receiver.

Then, the asCollector combinator wraps arguments into a collecting array. Here, we bind four arguments into an array of Object instances. Using the code in **Listing 8**, we can check that the resulting method handle is of type (Object,Object,Object,Object)void and that invoking it with arguments of the expected arity works.

Performance-wise, you should use invokeExact rather than invokeWithArguments whenever you can. invokeExact requires the argument types to be exactly like those of the method handle type descriptor, and it dispatches the call much faster. In contrast, invokeWithArguments performs type checks and conversions.

A classic optimization for virtual methods is to build *inline caches* where the target method handle is cached as long as the receiver type remains stable. You should limit the cache depth in case the call site happens to be megamorphic; otherwise, you would create too many method handles arranged in a chain of guarded tests, which further degrades the performance figures.

Bootstrapping

At this point, you should have a basic understanding of what method handles are. We can now turn to the bootstrapping side of invokedynamic.

CallSite objects. Call sites need to be bound at runtime using the java.lang.invoke API. To do that, each invokedynamic instruction refers to a *bootstrap method* that is executed the first time the call site invocation is executed.

The bootstrap method must return an instance of java.lang.invoke.CallSite. As the name suggests, such objects represent a call site, and they are bound to a target method handle, which may, in turn, be a chain of combinators.

The CallSite can be thought of as the container for the MethodHandle. Once it has been bound, a call site always references the same CallSite instance. Although the CallSite itself will refer to the same program point throughout its lifecycle, call sites may allow their target method handle to be redefined, providing a way to swap out old code with new code on the fly, without regenerating the bytecode for an entire method.

The java.lang.invoke package offers three concrete implementations of the abstract CallSite class, and you can subclass them, too:

- ConstantCallSite is for call sites whose target method handle never changes.
- MutableCallSite has object field semantics, which means that the target method handle can be changed.
- VolatileCallSite is similar to MutableCallSite but with volatile reference semantics.

If you need a call site where the target can be changed, choosing between MutableCallSite and VolatileCallSite is a matter of understanding the Java memory model and weighing the consequences of choosing one versus the other. In multithreaded environments, a *volatile call site target*

LISTING 4 / LISTING 5 / LISTING 6 / LISTING 7 / LISTING 8

```
static void printAll(String prefix, Object... values) {
 for (Object value : values) {
  System.out.println(prefix + value);
 }
}
```

Download all listings in this issue as text

change will be immediately visible to all threads, while with ordinary field semantics, thread caching occurs and requires explicit synchronization (see the static syncAll method in MutableCallSite). Of course, there are performance implications, too.

Bootstrap methods. invokedynamic bootstrap instructions are bound by invoking a static method that returns a CallSite object that must take at least three parameters:

- A MethodHandles.Lookup object that is bound to the classes visible in context of the call site
- A symbolic name as a string
- A MethodType that corresponds to the type that is expected by the call site (this is often useful for performing a final asType() combinator invocation to ensure that method handles match)

Other parameters can be passed as extra arguments, with the only constraint being that they must be constant values, which means they can be written to the *constant pool* of a class bytecode representation. This is the case for primitive types, strings, class references, and method handles.

KEY FOR LAMBDAS

Java 8 is poised to take advantage of invokedynamic as a way to support lambdas.

Sticking to the example in **Listing 8**, which obtained a method handle for the printAll method, the code in **Listing 9** would be a possible call site bootstrap method.

This method returns a nonmodifiable call site, where the prefix string is bound to ">>>" and the remaining arguments are collected as variable arguments. The resulting call site is of type (Object[])void. Note that because a string is a constant, we could also obtain the prefix as an extra parameter, and the value would depend on what is referenced from the call site in the bytecode. In this case, the bootstrap method would be as shown in **Listing 10**.

It is easy to test the bootstrap method to verify that it works as intended. Indeed, call sites provide a dynamicInvoker method that returns a method handle. Thus, you can use it as we've previously seen (see **Listing 11**).

Emitting Bytecode with invokedynamic Instructions

We are now ready to assemble the final pieces and actually put invokedynamic into practice. While the java.lang.invoke API can be

LISTING 9 LISTING 10 / LISTING 11 / LISTING 12

```
public static CallSite bootstrap(Lookup lookup, String name,
                    MethodType type) throws Throwable {
  MethodHandle target = lookup.findStatic(
    Sample3.class, "printAll",
      methodType(void.class, String.class, Object[].class))
      .bindTo(">>> ")
      .asVarargsCollector(Object[].class)
      .asType(type);
  return new ConstantCallSite(target);
}
```

Download all listings in this issue as text

used as is, it remains to be seen how it can be leveraged directly from bytecode instructions.

As of Java 7, there is *no* language construct that translates into invokedynamic instructions at the bytecode level. This might change in Java 8 because some of the support for lambdas is likely to be implemented with invokedynamic.

In order to bootstrap a call site and leverage a method handle, we need to use a third-party library to write bytecode. The ASM library is our weapon of choice, because it helps in reading, writing, and transforming bytecode.

Another option for simpler cases is to use a tool such as Indify that can replace some marked call sites with an invokedynamic call, but leveraging ASM leaves no magic behind.

Let's consider the code in **Listing 12**.

The code generates a sample3.Caller class as a subclass of java.lang.Object. It has a main method that loads two strings and then calls a console:print function of type (Object, Object)void that is bootstrapped using the static method that we defined earlier. The symbolic name for the function can be anything, but it is common to normalize names and separate portions using a colon, as in property:get, property:set, and so on.

We can run this method, as shown in **Listing 13**, and see that the invokedynamic instruction is correctly bound at runtime. We can also check the generated bytecode with the javap decompiler that comes as part of the JDK, as shown in **Listing 14**.

BETTER SUPPORT

The introduction of invokedynamic to the JVM specification is recognition of the fact that dynamic languages bring value to the Java ecosystem and, as such, the JVM needs to provide better support for them.

Conclusion

This article introduced invokedynamic, a new instruction backed with runtime API support for facilitating the implementation and execution of dynamic languages on top of the JVM. The java.lang.invoke API provides a rich set of operations to adapt a call site to a target that might be of a different signature type.

Existing dynamic languages for the JVM (for example, Groovy and JRuby) are starting to support invokedynamic. In the long run, this should reduce the footprint of their runtime support code because maintaining Java 6 backward compatibility will become less of an issue. Fresh language implementations, such as Oracle's Nashorn (JavaScript), Rémi Forax' PHP.reboot, or the present author's yet-to-be-released Golo language, are based on invokedynamic from the get-go. Java 8 is poised to take advantage of invokedynamic as a way to efficiently implement the support of lambdas. Finally, derivative usages—such as the JooFlux research project, which provides dynamic code replacement and aspect-oriented programming—are starting to appear.

While invokedynamic is mostly useful to language and middleware implementers, let nothing restrain your creativity! Are you thinking of derivative usages? Are you thinking about creating yet another language? In any case, there is no harm in trying, and who knows what you might come up with.

Acknowledgements. The author would like to thank Marcus Lagergren for his very constructive feedback. `</article>`

LISTING 13 **LISTING 14**

```
$ java -classpath code/build/classes sample3.Caller
>>> World
>>> Hello
```

Download all listings in this issue as text

LEARN MORE

- "Bytecodes Meet Combinators: invokedynamic on the JVM"
- "JooFlux: Hijacking Java 7 InvokeDynamic to Support Live Code Modifications"
- "JSR 292 Cookbook"
- "Dynalink: Dynamic Linker Framework for Languages on the JVM"

RAOUL-GABRIEL URMA AND JONATHAN GIBBONS

BIO

Java Compiler Plug-ins in Java 8

Use a new plug-in mechanism to extend the Java compiler with new behavior.

Java 8 will bring a new mechanism that allows you to write plug-ins for the Java compiler (javac). A compiler plug-in lets you add new phases to javac without making changes to its code base. New behavior can be encapsulated in a plug-in and distributed for other people to use. For example, javac plug-ins could be used to do the following:

- Add extra compile-time checks
- Add code transformations
- Perform customized analysis of source code

Note: The API for creating javac plug-ins is still experimental for JDK 8; it is scheduled to ship in 2013.

In this article, we show how you can write a simple, customized source code analysis tool so you can learn how to leverage the plug-in mechanisms for your own applications. We find code patterns that check whether the result of calling get() on an object that is a subtype of Map is null. In other words we are looking for patterns such as the following, where expr is a subtype of java.util.Map.

```
expr.get(key) == null
```

This pattern could be contained within a conditional expression, a return statement, and so on. It should be reported in all cases.

How Do Plug-ins Differ from Annotation Processors?

Here are some ways that plug-ins differ from annotation processors:

- Plug-ins are more flexible. They can run at various points in the compilation pipeline through a TaskListener.
- Plug-ins do not use the model of processing rounds, which incurs additional overhead.
- Plug-ins have a simpler interface.
- Annotation processors are defined by a standard (JSR 269). Plug-ins are javac-specific.
- Plug-ins require the use of a ServiceLoader.

Java Compiler Plug-in Architecture

A javac plug-in supports two methods:

- First, a getName() method that returns the name of the plug-in for identification purposes
- Second, a call() method that is invoked by the javac with the current environment it is processing

The call() method gives access to the compiler functionalities through a JavacTask object, which allows you to perform parsing, type checking, and compilation. In addition, the JavacTask object lets you add observers (which are instances of TaskListener) to various events generated during compilation. This is done through the method addTaskListener(), which accepts an object TaskListener.

Listing 1 shows com.sun.source.util.Plugin, which uses the getName() and call() methods. **Listing 2** shows some methods available in com.sun.source.util.JavacTask, and **Listing 3** shows the methods available in com.sun.source.util.TaskListener.

But there are obvious questions that remain unanswered, for example, how do you start writing the compiler plug-in and how do you run it?

Let's Build a Compiler Plug-In

The first step is to download and build the current release of the Java 8 compiler, which supports compiler plug-ins. Toward this end, download the latest version and fol-

low the build instructions in the README file.

Once the compiler is built, include the generated dist/lib/classes.jar file in your project. Alternatively, you can download a ready-made binary from jdk8.java.net.

Next, there are several steps we need to follow to build our plug-in. Here is a summary of the steps:

1. Implement the com.sun.source.util.Plugin interface shown in **Listing 1**.
2. Add a TaskListener to perform additional behavior after the type-checking phase.
3. Create an abstract syntax tree (AST) visitor to locate binary expressions:
   a. Evaluate whether the left side is a method call expression with a receiver's type that is a subtype of java.util.Map.
   b. Evaluate whether the right side is a null expression.

Step 1: Implement the com.sun.source.util.Plugin interface. The first step is to create the main class, which implements com.sun.source.util.Plugin. We return the name of our plug-in via the getName() method, as shown in **Listing 4**.

JUST PLUG IN

Plug-ins are flexible, have a simple interface, can run at various points in the compilation pipeline, and are specific to javac.

Step 2: Add a TaskListener. We are looking for a code pattern that checks whether the receiver of the method call get() is a subtype of java.util.Map. To get the type of the receiver, we need information about the types of expressions that are resolved during the type-checking phase. We therefore need to insert a new phase after type checking. Toward this end, we first create a TaskListener and add it to the current JavacTask, as shown in **Listing 5**.

We now need to create the class CodePatternTaskListener, which implements a TaskListener. A TaskListener has two methods, started() and finished(), which are called, respectively, before and after certain events. These events are encapsulated in a TaskEvent object. In our case, all we need to do is implement the finished() method and check for an event mentioning the *Analyze* phase (type checking), as shown in **Listing 6**.

Step 3: Create the AST visitor. Next, we need to write the logic to locate the code pattern and report it. How do we do that? Thankfully, a task event provides us with a CompilationUnitTree, which represents the current source file analyzed in a tree structure. It can be accessed with the getCompilationUnit() method.

Our code will need to traverse this tree, locate a binary node, and evaluate the node's left and right children. This sounds like a visitor pattern job. The Compiler Tree API provides us with a ready-made visitor class designed for such tasks: com.sun.source.util.TreeScanner<R, P>. It visits all the nodes in the AST.

First, we initialize our visitor object and then visit the CompilationUnitTree:, as shown in **Listing 7**.

All we have left to do is to override visitBinary(BinaryTree node, P p) and write the logic to verify the code pattern. The full code of the visitor class with inlined comments is provided in **Listings 8a–8d**.

LISTING 1 | LISTING 2 | LISTING 3 | LISTING 4 | LISTING 5 | LISTING 6

```
public interface Plugin {
    public String getName();
    public void call(JavacTask task, String[] pluginArgs);
}
```

Download all listings in this issue as text

Let's Run Our Compiler Plug-In

We are almost finished. The final step is to set up a file called com.sun.source.util.Plugin located in META-INF/services/. This file

must contain the name of our plug-in, CodePatternPlugin, which allows javac to load the appropriate plug-in.

Next, using your favorite IDE or the command line, create a Java archive (JAR) file from your project containing the META-INF directory and compiled class files:

```
$ jar –cvf codePatternPlugin.jar
    META-INF *.class
```

Finally, you can run the plug-in as shown in **Listing 9**, where

- -processorpath indicates the path where the plug-in JAR file is located
- -Xplugin indicates the name of the plug-in to run, which is CodePatternPlugin, in this case

Conclusion

The new plug-in mechanism provides a simple hook to javac. You can use it to extend javac with new behavior. In addition, you can distribute a plug-in without making modifications to the javac code base. In this article, we showed how you can use this mechanism to easily write a customized source code analysis tool for your applications. `</article>`

LEARN MORE

- Complete project and source files on github

LISTING 7 / LISTING 8a / LISTING 8b / LISTING 8c / LISTING 8d / LISTING 9

```
if(taskEvent.getKind().equals(TaskEvent.Kind.ANALYZE))
{
  CompilationUnitTree compilationUnit =
   taskEvent.getCompilationUnit();
  new CodePatternTreeVisitor().scan(compilationUnit, null);
}
```

Download all listings in this issue as text

The New javax.cache Caching Standard

Learn how to use javax.cache.

GREG LUCK

BIO

The javax.cache standard being developed by JSR 107 aims to standardize the use of caching in Java. Why? Open source caching projects and commercial caching vendors have been available for more than a decade. The distributed kind, which is often called a *distributed cache*, has entered wide adoption. While each vendor uses a very similar map-like API for basic storage and retrieval, each vendor uses its own API.

At present, developers will incur a significant coding cost in changing from one proprietary implementation to another. It is also challenging for independent software vendors (ISVs) to introduce pluggable caching to support multiple implementations. Like developers, ISVs either create their own service provider interface or they support just a few implementations—a situation that isn't great for anyone. Other areas of Java, such as JDBC, the Java Persistence API (JPA), and Java Message Service (JMS), have solved this problem through standards.

Indeed, the analyst firm Gartner recently reported that the lack of a standard in this area was the single biggest inhibitor to mass adoption of data grids. At the time of this writing, JSR 107 was in Early Draft review and expected to reach final approval toward the end of 2012.

The code listings in this article work with the Early Draft version of the API (0.5).

About Caching

While caching seems like a simple topic, there are subtleties. A cache is a place where you put a copy of data that is intended to be used multiple times. Caching implementations, being in memory, are much faster than the original source of the content. This means you get a performance benefit from using a cache instead of the original source. You also offload the resource that the original data came from.

IT'S SUBTLE
While caching seems like a simple topic, **there are subtleties.**

To be effective, data needs to be used multiple times. There is no value in caching data that is written once and never read or read only once. The efficiency of the cache can be measured by the *hit ratio*, which is defined as cache hits/cache requests.

To provide the maximum offload, caches need to be distributed so that the work performed by one application server gives benefit to the others and eliminates any duplicate requests for the same data to the underlying resource.

Finally, the affordability of servers with memory capacities of 1 TB and higher, combined with vendor innovation to utilize that memory for cache storage, resulted in a new trend in which the cache has increased operational significance. Instead of caching just part of a data set, the entire data set is placed in cache and is used as an authoritative source of information—the cache, in essence, becomes the operational store for the application. In this use

case, the cache is often referred to as a *data grid*.

Each of these areas has requirements that the standard must deal with.

Who Is Adopting javax.cache?

At the time of this writing, JSR 107 is planned to be included in Java EE 7 and developed by JSR 342. Java EE 7 is due to be finalized in 2013. In the meantime, javax.cache will work in Java SE 6 and higher and in Java EE 6 environments as well as with Spring and other popular environments.

The following vendors are either active members of the Expert Group or have expressed interest in implementing the specification:

- Terracotta (Ehcache)
- Oracle (Oracle Coherence)
- JBoss (Infinispan)
- IBM (ExtremeScale)
- SpringSource (Gemfire)
- GridGain
- TMax
- Fujitsu (Interstage XTP)

Terracotta, Oracle, and JBoss are all planning releases upon finalization of the spec. In addition, Spring plans to support JSR 107 closer to its final release.

Main Features

From a design point of view, the basic concept is a CacheManager that holds and controls a collection of Cache instances that have entries. The API can be thought of as maplike with the following additional features:

- Atomic operations, similar to java.util.ConcurrentMap
- Read-through caching
- Write-through caching
- Cache event listeners
- Statistics
- Transactions including all isolation levels
- Caching annotations
- Generic caches that hold a defined key and value type
- Support for storage by reference (applicable to on-heap caches only) and storage by value

Optional Features

Rather than split the specification into a number of editions targeted at different user constituencies, such as Java SE or Spring and Java EE, we have taken a different approach.

First, for Java SE–style caching, there are no dependencies. And for Spring and Java EE, where you might want to use annotations and transactions, the dependencies will be satisfied by those frameworks.

Second, we have a capabilities API via ServiceProvider.isSupported(OptionalFeature feature) so you can determine at runtime the capabilities of the implementation.Optional features include the following:

- storeByReference (storeByValue is the default)
- Transactions
- Annotations

This makes it possible for an implementation to support the specification without necessarily supporting all the features, and it allows end users and frameworks to discover the features so they can dynamically configure appropriate usage.

LISTING 1 | LISTING 2 | LISTING 3 | LISTING 4 | LISTING 5 | LISTING 6

```
<dependency>
  <groupId>javax.cache</groupId>
  <artifactId>cache-api</artifactId>
  <version>0.5</version>
</dependency>
```

Download all listings in this issue as text

Good for Simple Caches to Data Grids

While the specification does not mandate a particular distributed topology, it is cognizant of the fact that data storage might be distributed. We have one API that covers both usages but is sensitive to distributed concerns. We do not have high-network-cost maplike methods, such as keySet() and values(). And we generally prefer zero or low-cost return types. So while Map has V put(K key, V value), javax.cache.Cache has void put (K key, V value).

Class Loading

Caches contain data shared by multiple threads, which might themselves be running in different container applications or Open Services Gateway initiative (OSGi) bundles within one Java Virtual Machine (JVM) and might be distributed across multiple JVMs in a cluster. This makes class loading tricky.

We have addressed this problem. When a CacheManager is created, a class loader can be specified. If none is specified, the implementation provides a default. In either case, object deserialization will use the CacheManager instance's class loader.

This is a big improvement over the approach taken by caches

such as Ehcache that use a fall-back approach. With a fall-back approach, first, the thread's context class loader is used, and if that fails, another class loader is tried. This can be made to work in most scenarios but is a bit hit-and-miss and varies considerably by implementation.

Getting the Code

The spec is in Maven central. The Maven snippet is shown in **Listing 1**.

A Cook's Tour of the API: Creating a CacheManager

We support the Java 6 java.util.ServiceLoader creational approach, which will automatically detect a cache implementation in your classpath. You then create a CacheManager with the code shown in **Listing 2**, which returns a singleton CacheManager called Wdefault_. Subsequent calls return the same CacheManager.

CACHE THIS

The API supports programmatic creation of caches. This complements the usual convention of configuring caches declaratively.

CacheManager instances can have names and class loaders configured, for example, as shown in **Listing 3**. Implementations can also support direct creation with new for maximum flexibility, as shown in **Listing 4**.

Or, to do the same thing without adding a compile-time dependency on any particular implementation, use the code shown in **Listing 5**.

We expect implementations to have their own well-known configuration files, which will be used to configure the CacheManager. The name of the CacheManager can be used to distinguish the configuration file. For Ehcache, this will be the familiar ehcache.xml placed at the root of the classpath with a hyphenated prefix for the name of the CacheManager. So, the default CacheManager will simply be ehcache.xml and myCacheManager will be app1-ehcache.xml.

Creating a cache. The API supports programmatic creation of caches. This complements the usual convention of configuring caches declaratively, which is left to each vendor.

To programmatically configure a cache named testCache, which is set for read-through, use the code shown in **Listing 6**.

Getting a reference to a cache. You get caches from the CacheManager. To get a cache called testCache, use the code shown in **Listing 7**.

LISTING 7 LISTING 8 / LISTING 9 / LISTING 10 / LISTING 11

```
Cache<Integer, Date> cache = cacheManager.getCache("testCache");
```

Download all listings in this issue as text

Basic cache operations. To put data in a cache, use the code shown in **Listing 8**. To get data from a cache, use the code shown in **Listing 9**. To remove data from a cache, use the code shown in **Listing 10**.

Annotations

JSR 107 introduces a standardized set of caching annotations, which perform method-level caching interception on annotated classes running in dependency injection containers. Caching annotations are becoming increasingly popular, starting with Ehcache Annotations for Spring, which influenced Spring 3's caching annotations. Spring plans on supporting the new standard annotations in addition to its own.

The JSR 107 annotations cover the most-common cache operations, including the following:

- @CacheResult—Use the cache.
- @CachePut—Put into the cache.
- @CacheRemoveEntry—Remove a single entry from the cache.
- @CacheRemoveAll—Remove all entries from the cache.

When the required cache name, key, and value can be input, they are not required. See the Javadoc for the details. To allow greater control, you can specify all these operations and more.

In the example shown in **Listing 11**, the cacheName attribute is specified to be domainCache, index is specified as the key, and domain is specified as the value.

The Reference Implementation includes an implementation for Guice, Spring, and Contexts and Dependency Injection (CDI). CDI is the standardized container-driven injection introduced in Java EE 6. The implementation is nicely modularized for reuse and uses

an Apache license; therefore, we expect several open source caches to reuse the implementation.

Annotation example. The example in **Listing 12** shows how to use annotations to keep a cache in sync with an underlying data structure, in this case a blog manager, and also how to use the cache to speed up responses, done with @CacheResult.

Wiring up Spring. For Spring, the key is the configuration line shown in **Listing 13**, which adds the caching annotation interceptors into the Spring context. **Listing 14** shows a full example.

Spring has its own caching annotations based on earlier work from JSR 107 contributor Eric Dalquist. Those annotations and JSR 107 will happily coexist.

Wiring up CDI. First, create an implementation of javax.cache.annotation.BeanProvider and then instruct CDI where to find it by declaring a resource named javax.cache.annotation.BeanProvider in the classpath at /META-INF/services/.

For an example using the Weld implementation of CDI, refer to the CdiBeanProvider in our CDI test harness.

Conclusion

The javax.cache specification standardizes a long-overdue area of Java. A rich variety of commercial and open source implementations will be available, which should ensure very broad adoption. The standard accommodates simple in-process caches but will also provide the primary API for storage into and retrieval from data grids.

Those with a need to use a cache or a data grid should look to adopt the specification upon its release to avoid vendor lock-in. Because the standard works with Java SE, Java EE 6 and 7, and other frameworks such as Guice and Spring, enterprises should consider using it for future projects. `</article>`

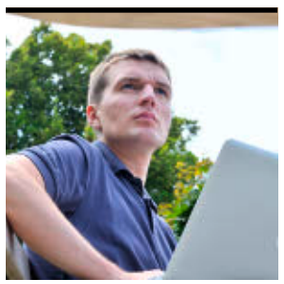
LISTING 12 LISTING 13 / LISTING 14

```
public class BlogManager {

  @CacheResult(cacheName="blogManager")
  public Blog getBlogEntry(String title) {...}

  @CacheRemoveEntry(cacheName="blogManager")
  public void removeBlogEntry(String title) {...}

  @CacheRemoveAll(cacheName="blogManager")
  public void removeAllBlogs() {...}

  @CachePut(cacheName="blogManager")
  public void createEntry(@CacheKeyParam String title,
      @CacheValue Blog blog) {...}

  @CacheResult(cacheName="blogManager")
  public Blog getEntryCached(String randomArg,
    @CacheKeyParam String title){...}

}
```

Download all listings in this issue as text

LEARN MORE

- JSR 107 home page
- JSR 107 status



ADAM BIEN

Secure Java EE Authentication

Implement login authentication using declarative and programmatic security.

Java EE security is easy, powerful, extensible, and sufficient for the vast majority of all use cases. This article discusses RESTful services secured with standard Java EE capabilities, and it demonstrates a domain-specific extension implemented with Contexts and Dependency Injection (CDI) and Enterprise JavaBeans (EJB) 3.

Instead of covering all authorization and authentication aspects—which is impossible in a single article—this article focuses on the most-common use cases: login authentication methods with declarative and programmatic security. **Note:** Some relevant example code can be found here.

Setting the Stage

For the purposes of this article, we will implement a system that, by default, displays some "wisdom"—the message "Java Programming Language Rocks!"—and allows a chosen group of people called "dukes" to change the message.

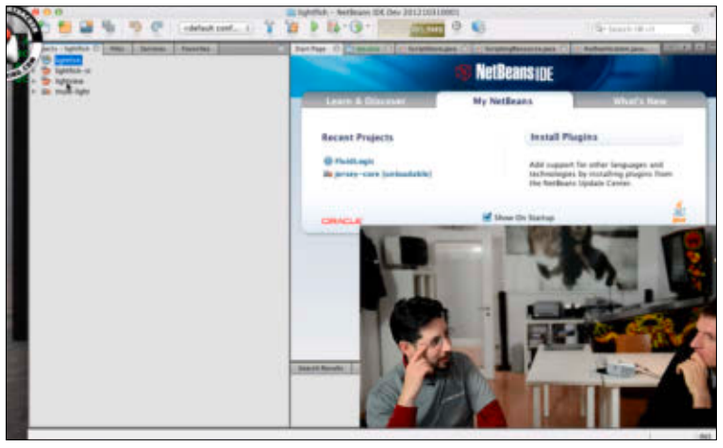
All HTTP-GET requests of the wisdom URI should be available to everyone, whereas only members of the dukes group should be able to execute the @POST method shown in **Listing 1**.

REST resource implementations are usually heavily dependent on the JAX-RS API, which encourages the introduction of an explicit managed bean responsible for the message management. The WindowStorage bean is declaratively secured with @RolesAllowed annotations (see **Listing 2**).

The parameterless wisdom method can be executed by anyone, but only a user in the dukes role is allowed to change wisdom by passing a string as parameter. All used roles have to first be declared on the class level with the @DeclareRoles annotation. You could also use an XML deployment descriptor for this purpose. The @PermitAll annotation allows all users to execute an annotated method, and the @RolesAllowed annotation restricts access to the specified roles.

Authentication as Aspect

So far, the user is not going to be authenticated. Without authentication, the current user is not going to be identified as a Principal with its configured roles membership. In the case of GlassFish, a nonauthenticated user is associated with a Principal with the ANONYMOUS name. Fortunately, most of the Web frameworks, and so also the JAX-RS API, are Servlet-based. Servlets come with an easy way to authenticate a user with a single invocation of the method HttpServlet Request#authenticate. The entire procedure can be



Adam Bien talks with Oracle's Stephen Chin about his current projects as part of Chin's NightHacking Tour across Europe.

PHOTOGRAPH BY THOMAS EINBERGER/ GETTY IMAGES

extracted into a reusable Web filter, as shown in **Listing 3**.

With the annotation @WebFilter("/*"), the filter intercepts all requests. The crux of the Authenticator filter is the invocation of the authenticate method.

Where Annotations End and XML Starts

In the old Java EE tradition, no single XML line has been written so far. Unfortunately, we will have to specify a few lines in web.xml (see **Listing 4**) to configure which authentication method will be used and from which realm the authentication and authorization information is fetched.

The error-page section is needed only to display for the user a nicer message than an "Internal Server Error 500" message. Because of its flexibility, form-based login configuration is the most popular. To pass the username and password to the system, you only have to use j_security_check as a name of the action and j_username and j_password to pass the username and password, respectively. In our example, the form is implemented as a static HTML page, as shown in **Listing 5**.

Of course, you could equally well pass the information directly with the POST call programmatically. You are not limited to using a static HTML page to perform the authentication.

What Is file?

Within the web.xml deployment descriptor, a realm with the name file is referenced. A realm is a user repository containing the security credentials as well as the roles membership. A concrete realm realization is not specified by the Java EE specification. Usually, application servers tend to support file, database, and LDAP realms out of the box, and they support a large number of extensions to cover a wide variety of legacy resources. For demo purposes, we'll later create a file realm with the name file and a user with the name james and membership to the dukes role.

As an enterprise developer, you cannot fully rely on concrete names existing in the production repository. Instead of hardcoding volatile role names into your code, you can use fictive role names and map them to the actual ones in the deployment descriptor.

Logical roles that come with the application are called *groups*. The Java EE spec foresees the mapping of roles to groups in the deployment descriptor. Of course, such a decoupling works only in cases where roles and groups with different names are semantically equivalent. If you can influence the actual role naming in the target realm, you can also conventionally use the roles as groups and omit the mapping. In the case of GlassFish, an automatic role-to-group mapping can be specified directly in the security realm configuration.

LISTING 1 LISTING 2 LISTING 3 LISTING 4 LISTING 5

```
@Stateless
@Path("wisdom")
@Produces(" text/plain") // MediaType.TEXT_PLAIN
public class WisdomResource {

  @Inject
  WisdomStorage storage;

  @GET
  public String wisdom() {
    return storage.wisdom();
  }

  @POST
  @Consumes(MediaType.APPLICATION_FORM_URLENCODED)
  @Produces(MediaType.TEXT_HTML)
  public String wisdom(
      @FormParam("wisdom") String wisdom) {
    storage.wisdom(wisdom);
    return "thanks!";
  }
}
```

Download all listings in this issue as text

Who Is Responsible for the 401 Error?

The error-page section in web.xml specifies a redirect to an unauthorized .html page upon any occurrence of the HTTP 401 error. Although such a centralized redirect is useful and nice, the HTTP 401 error will never happen in our sample code.

Every serious blogger would love to change the "Java Programming Language Rocks!" wisdom to something like "Java Is the New COBOL!" Any attempt to do so would result in error 500 with output like the following, but it would never result in error 401:

```
javax.servlet.ServletException:
javax.ejb.AccessLocalException:
Client not authorized for this
invocation
```

Fortunately, JAX-RS comes with a clean way to map any exceptions into sensible HTTP error codes. The AccessLocalExceptionMapper class is responsible for the conversion of any AccessLocalException into the 401 error code. On each occurrence of AccessLocalException, the active user is also logged out (see **Listing 6**).

Beyond marking the ExceptionMapper implementation with the @Provider annotation, no further configuration is needed. An ExceptionMapper implementation will be automatically discovered and activated.

When Declarative Is Not Enough

So far, all role names were bound to the methods declaratively. Every declaration change requires a full recompilation and redeployment. Instead of hardcoding the role names in the annotations, they can be accessed programmatically. With an injected SessionContext, the role membership can be verified at runtime with the SessionContext#isCallerInRole invocation. The Principal instance representing the current user can be directly injected into any managed bean that allows implementation of a finer-grained authorization, as shown in **Listing 7**.

In contrast to the declarative approach, the role can be chosen at runtime from the set of declared roles and changed without recompilation. However, the permission management has to be directly performed in the security realm and might be hard to implement in the real world.

Producing More Flexibility on the Fly

The Principal name is usually a username, which in turn is used as user identification. Because of the unique user identification requirement, the username can also be used as a unique identifier. Any arbitrary information from a custom permission store can be associated with the username, fetched from a custom realm implementation, and evaluated at runtime. Usually, the additional security information is closely related to the target domain. There is no significant domain complex-

LISTING 6 LISTING 7

```
@Provider
public class AccessLocalExceptionMapper implements
    ExceptionMapper<AccessLocalException> {
 @Context HttpServletRequest request;

 @Override
 public Response toResponse(AccessLocalException exception) {
   try {
    request.logout();
   } catch (ServletException ex) {
   }
  return Response.status(Response.Status.UNAUTHORIZED).build();
 }
}
```

Download all listings in this issue as text

ity in our example, so we will use only generic permissions:

```
public enum Permission {
  READ,WRITE,EXECUTE;
}
```

In the real world, you can attach to a Principal whatever is beneficial for the realization of your requirements—it doesn't have to be a Permission. It can be any flat or hierarchical data structure. Usually the custom permissions are going to be maintained in a persistent store separately. For demonstration purposes, the Permission instances are maintained in an in-memory custom realm implemented as a singleton EJB bean, as shown in **Listing 8**.

In the real world, you could extend the InMemoryPermissionsRealm to access the persistent store and cache the results. For reasons of convenience, the generic set of Permissions is wrapped with a more meaningful class representing the user, as shown in **Listing 9**.

KILLER USE CASE

Authentication and authorization are one of the few killer use cases that can be solved with aspects.

A user populated with a custom set of permissions needs to be conveniently exposed to the application code in order to be useful. For the population of the user instance with the externally stored permissions, the name of the currently active user is needed. The class UserProvider uses the username stored in the injected Principal instance to instantiate a User with the Permission instances from the InMemoryPermissionsRealm (see **Listing 10**).

Now a user instance with custom permissions can be conveniently injected into any Java EE component:

```
public class SomeComponent {
  @Inject
  User currentUser;
}
```

Declarative Authorization with Even More Aspects

Although a User instance populated with custom permissions can be injected to all instances participating in the call stack, the handling is still insufficiently convenient. You would have to ask the User for authorization each time you are planning to invoke a guarded method. However, authentication and authorization are one of the few killer use cases that can be solved with aspects. Instead of repeatedly asking the User for permission, a guarded method can be denoted with a plain annotation, as shown in **Listing 11**.

The AllowedTo annotation carries a Permission array and can be directly applied on business methods (see **Listing 12**). The actual checks were factored out into an interceptor, which compares the

LISTING 8 / LISTING 9 / LISTING 10 / LISTING 11 / LISTING 12

```
@Singleton
@ConcurrencyManagement(ConcurrencyManagementType.BEAN)
public class InMemoryPermissionsRealm {

  private Map<String,EnumSet<Permission>> customStore;

  @PostConstruct
  public void populateRealm(){
    this.customStore =
   new HashMap<String, EnumSet<Permission>>();
    this.customStore.put("james",
   EnumSet.allOf(Permission.class));
    this.customStore.put("blogger",
   EnumSet.noneOf(Permission.class));
  }

  public EnumSet<Permission> getPermissionForPrincipal(
   String userName){
    EnumSet<Permission> configuredPermissions =
   this.customStore.get(userName);
    if(configuredPermissions != null) {
      return configuredPermissions;
    }
    else {
      return EnumSet.noneOf(Permission.class);
    }
  }
}
```

Download all listings in this issue as text

value of the Permission annotation with the permissions of the active user, as shown in **Listing 13**.

An interceptor has access not only to the methods but also to all parameters. In addition, the User is injectable to Java EE components. With custom permissions and direct access to the business logic, instance-based authorization can be easily realized. Usually, the challenges are not Java EE limitations, but rather a deep understanding of the target domain. While it is straightforward to give only specified users permission to access particular rows in the table, it is significantly harder to find an elegant abstraction in practice.

STRONG COMBO

The combination of built-in security with CDI and Servlet API capabilities allows a portable implementation of custom authorization schemes with only a few lines of code.

The Very Last Aspect

All attempts to access a guarded method without a valid permission result in a SecurityException, which will again cause an ugly "Internal Server Error 500" message. We could throw an AccessLocalException to reuse the already implemented AccessLocalExceptionMapper, but it would unnecessarily tighten the generic Guard to the EJB API. An additional ExceptionMapper implementation converts any SecurityException into an HTTP 401 error code and logs out the user, as shown in **Listing 14**.

The SecurityExceptionMapper class is a 1:1 copy of AccessLocalExceptionMapper, but it is still a slightly better solution than introducing a dependency on the EJB API. A JAX-RS ExceptionMapper is a great way to reduce the exception-handling plumbing inside the resource classes. Alternatively, a redirect to the unauthorized page could also be specified in the web.xml file based on the exception type, not on a status code (see **Listing 15**).

What's Left

Java EE 6 authorization and authentication comes with surprising power and flexibility. The vast majority of mainstream use cases can be solved efficiently out of the box without any extensions or libraries. However, third-party frameworks such as OpenAM, JBoss SSO, or Apache Shiro come with custom

LISTING 13 LISTING 14 / LISTING 15

```
public class Guard {

  @Inject
  Instance<User> users;

  @AroundInvoke
  public Object validatePermissions(
   InvocationContext ic) throws Exception{
     Method method = ic.getMethod();
     User user = users.get();
     if(!isAllowed(method, user)){
        throw new SecurityException("User " + user + " is "
   + "not allowed to execute the method " + method);
     }
     return ic.proceed();
  }

   boolean isAllowed(Method method,User u) {
     AllowedTo annotation = method.getAnnotation(
   AllowedTo.class);
     if(annotation == null) {
        return true;
     }
     Permission[] permissions = annotation.value();
     for (Permission permission : permissions) {
        if(u.isAllowed(permission)){
           return true;
        }
     }
     return false;
   }
}
```

Download all listings in this issue as text

realm implementations that offer connectivity to a variety of stores and APIs. The form-based authentication discussed here is the most popular, but it is not the only one available. Also, the insecure Basic, the slightly more secure Digest, and the secure but harder-to-deploy Certificate authentications are available but less common in practice. In particular, RESTful interfaces are heavily based on URIs and can be conveniently secured by restricting accessibility to a certain URI and method with the web-resource-collection tag in web.xml.

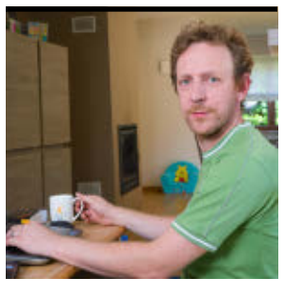
Conclusion

The combination of built-in security with CDI and Servlet API capabilities allows a portable implementation of custom authorization schemes with only a few lines of code. Even the way the application server resolves the roles and principals and enforces the rules is specified with the Java Authorization Contract for Containers (JACC) specification. JACC has been part of Java EE since J2EE 1.4 and has to be implemented by every certified application server.

You could completely change authentication and authorization with the implementation of the interface javax.security.jacc.PolicyConfiguration (14 methods) and the abstract class javax.security.jacc.PolicyConfigurationFactory (3 methods). Custom JACC implementations are often used to plug external authentication and authorization products in a portable way. `</article>`

LEARN MORE

- *Real World Java EE Patterns—Rethinking Best Practices*, "Re-Injector" and "Aspect" chapters (press.adam-bien.com, 2012)
- *Java Magazine* article "Who Needs Aspect-Oriented Programming?"
- "Contexts and Dependency Injection in Java EE 6"
- *Java Magazine* article "Convention over Configuration in Java EE 6"



JOHAN VOS

Integrating Web and Java Client Applications with Social Media

Learn how to integrate your application with existing social network systems.

Today, many IT projects need to integrate with social media, for example, Facebook and Twitter. Buttons with text such as "Log in via Twitter" or "Post to Facebook" are everywhere on the Web and beyond. While social media often present themselves as easy-to-use services to the end user, a deep integration at the API level requires consideration.

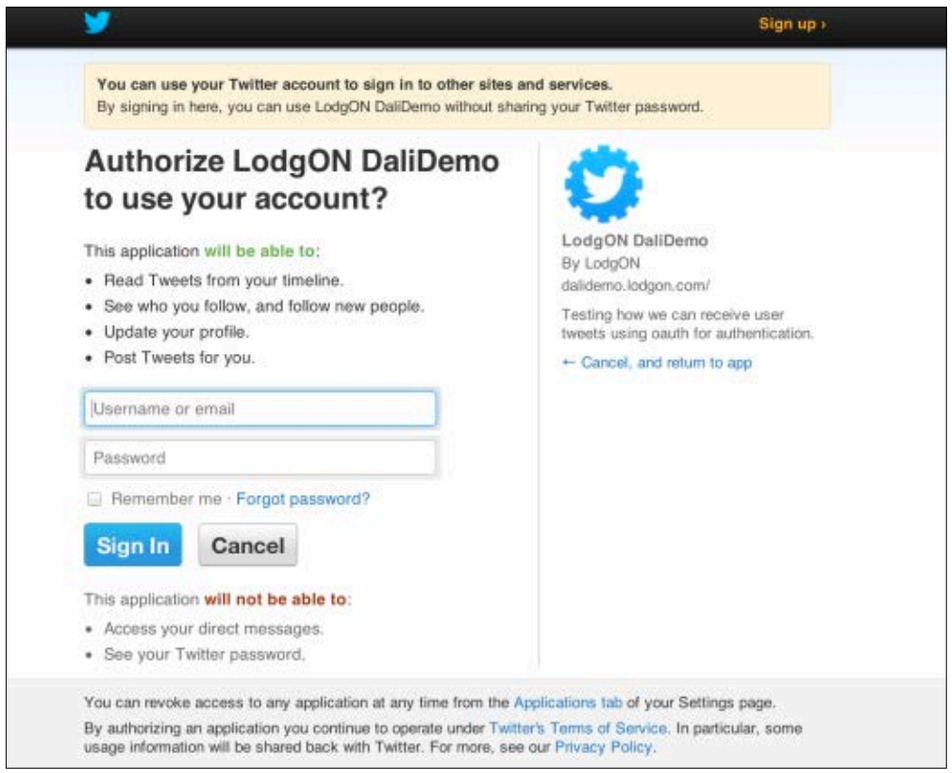


Figure 1

PHOTOGRAPH BY TON HENDRIKS

In this article, we show how you can easily integrate your Web and Java client applications with existing social media.

Social Media

There are a number of reasons why a Web or client application might benefit from integrating with existing social media:

- End users don't need to maintain additional credentials. Instead they can log in using their Facebook/Twitter credentials. It is a big hurdle when people have to create yet another account with yet another password for a new service.
- The "viral" aspect: Social media is about networking. People see what their peers are doing, and they might want to do the same.
- User information: Applications can benefit from additional user information that is already available in the user's profile on social media. Applications don't have to ask for this information—although the social network will ask users if they want to share this information with the application.

The level of integration varies among applications from easy to more challenging and might be described as follows:

- Public information, using anonymous access
- Authentication
- Read
- Write

Anonymous Access

Most social media provide limited, anonymous access to some services. As such,

integration with a social network is no different from integration with any other external Web service. Via a cURL command, the information can easily be retrieved using command-line requests, as shown in **Listing 1**.

The first call will retrieve tweets that are tagged with java and return the result in JSON format. The second call will return the public part of the profile of the Java Champions group on Facebook. A very basic Java program can be used to query Twitter and Facebook in order to retrieve information that is accessible without authentication, as shown in **Listing 2**.

Authentication

Typically, information that is available without a user having to log in on a Website is also available via an anonymous, nonauthenticated API call. Thus, this information can be retrieved easily by Java applications. It should be noted, though, that most social media sites impose volume restrictions. For example, you can make only 2,000 API calls per day or you can track only 10 users simultaneously.

Private—and often more interesting—information is accessible only by using authentication that involves three parties:

- The social network
- Your application
- The end user

Authenticated access to resources might seem more complex, but it comes with a number of benefits. The application knows the end user and has access to interesting information that can be valuable for its own purposes. To access user-specific information, the social network often requires that your application does this on behalf of the user. Typically, OAuth 1, OAuth 2, or a similar protocol is used for this. It is beyond the scope of this article to explain OAuth in depth. Interested readers can refer to the specification document for OAuth 1 and to the draft version of OAuth 2.

In general, an application is allowed to make private calls only when the end user allowed it (see **Figure 1**). This explains messages such as "this application wants to post on your timeline" and "this application wants to read your e-mail."

Rather than asking the user for permission for every single call, the application is often granted one user access token per user by a specific social network. All requests that the application makes on behalf of a specific user are then signed using that token. The social network then verifies whether the token provided by the application grants sufficient access rights to the requested resource, for example, to read the social graph or to post tweets.

Obtaining a user access token is a multistep process, which is described here for OAuth 1.0. In a browser application, users are redirected to a login page at the social network where they grant or deny permissions. The social network and the application communicate via a multiphase protocol, and the end result is the applica-

LISTING 1 LISTING 2

```
curl
   "http://api.twitter.com/1/users/lookup.json?screen_name=java"
curl "http://graph.facebook.com/JavaChampions"
```

Download all listings in this issue as text

tion obtaining a user access token. Users are then redirected from the application to the social network and back. Schematically, this is shown in **Figure 2**.

Before an application can request that users authenticate themselves with social media, the application needs to be registered with the social network. Every social network has its own registration process, but the following are a number of common concepts:

- The application needs to provide a callback URL that will be called by the social network upon successful authentication.
- The application is granted a consumerKey and a consumer Secret or a public key and a private key; terminologies tend to vary between different social media. This key combination is used when signing requests.

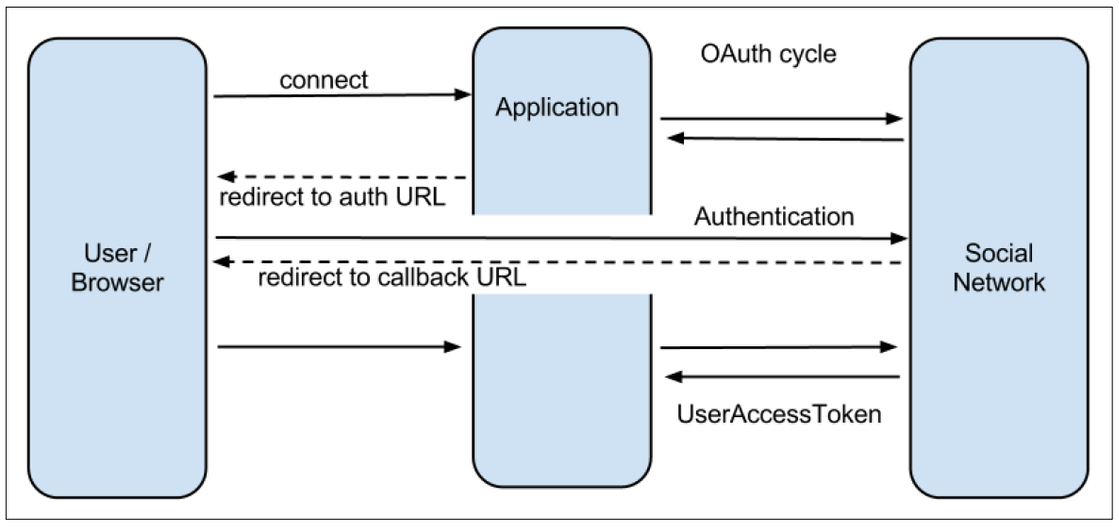


Figure 2

DaliCore

DaliCore is an open source project that adds the concepts of user, content, group, and permission on top of the Java EE 6 specification. The goal of DaliCore is to make it easier to develop applications that involve users and content, organize them in groups, and assign and check permissions. A high-level overview of the DaliCore architecture is shown in **Figure 3**.

DaliCore defines the concept of a user in the class com.lodgon.dali.core.entity.User. Operations on the user concept are defined in the stateless session bean com.lodgon.dali.core.ejb.UserBean.

The DaliCore project contains a module DaliCoreSocial, which is particularly useful in facilitating the above-mentioned process of obtaining user access tokens for different social media. By using DaliCoreSocial, an application developer is shielded from the rather complex flow that is required for authenticating a user with a social network.

A user of a social network corresponds to an instance of com.lodgon.dali.core.social.entity.OnlineAccount. A single user can be linked to zero or more OnlineAccount instances (for example, an OnlineAccount for Facebook and an OnlineAccount for Twitter).

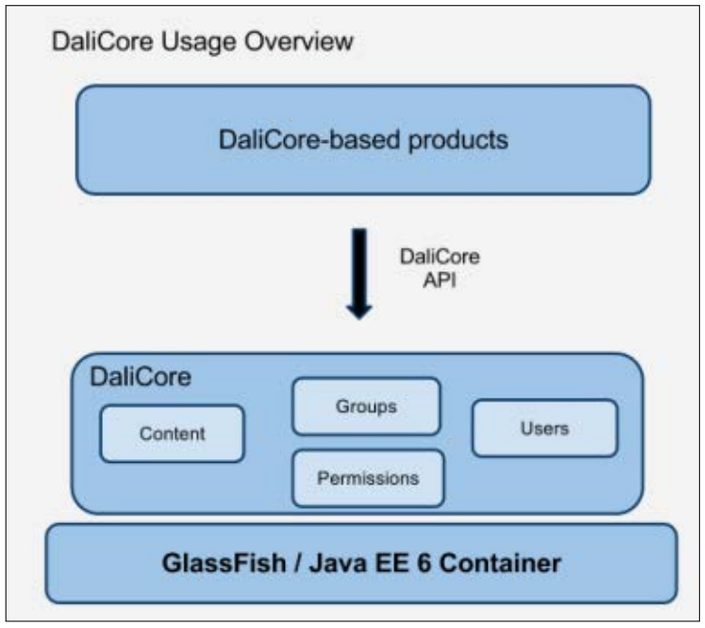


Figure 3

The business functionality provided by DaliCoreSocial is available in the stateless session bean com.lodgon.dali.core.social.ejb.SocialBean. This session bean contains functionality for finding which OnlineAccount instances are coupled with a particular User instance, for example:

- findOnlineAccount
- findOnlineAccountByUser

Also, DaliCoreSocial provides functionality that applications can use to retrieve information from a social network or to push information to a social network, for example:

- getFriends
- getStatus
- getStream
- postStatus

In order to couple an OnlineAccount to a User or create a new User with an OnlineAccount, the end user needs to connect with a social network. As mentioned before, an application needs to register itself with each of the social media it wants to communicate with. The registration process is specific to each social network. For example, registering your application with Twitter requires you to log in at http://dev.twitter.com and register the application, as shown in **Figure 4**.

After registering successfully with a social network, the application is granted a key and a secret. DaliCoreSocial requires these keys

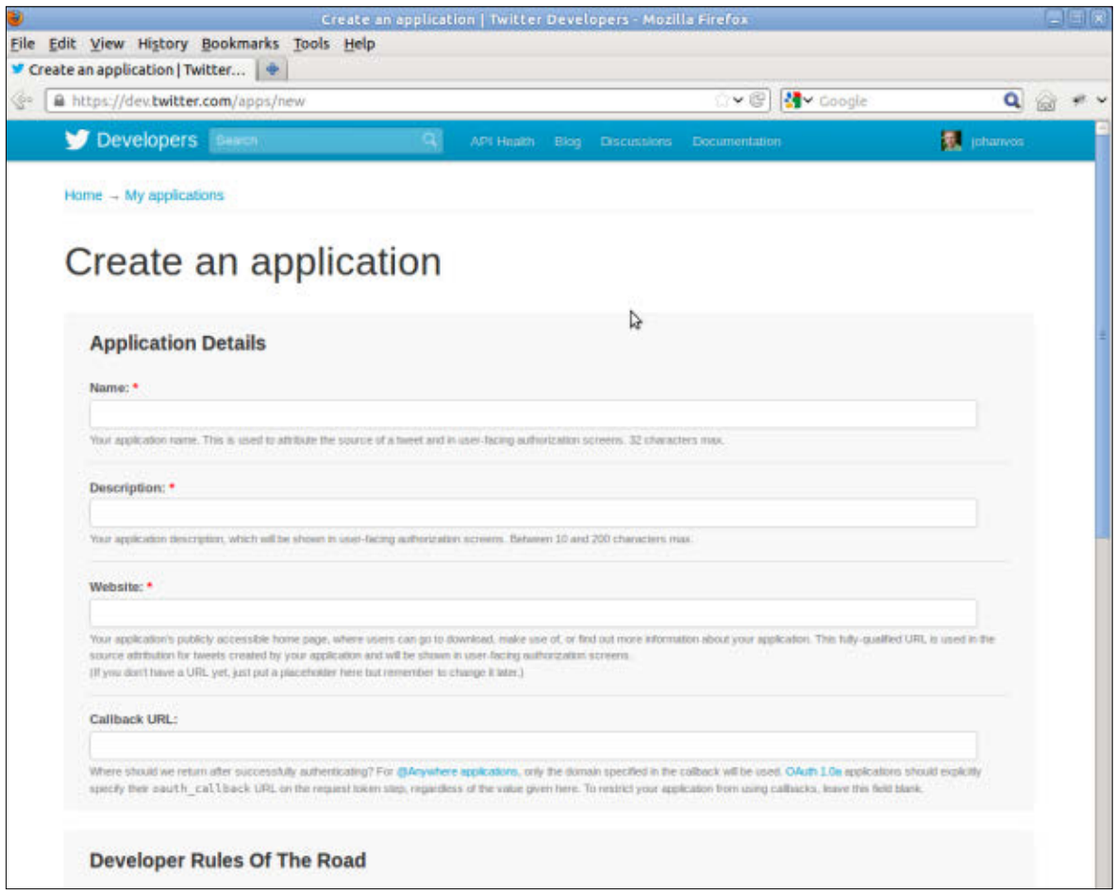


Figure 4

to be available in the web.xml file in the format shown in **Listing 3**.

DaliCoreSocial can be used by Web applications as well as by JavaFX applications, as shown in **Figure 5**.

If a Web application is required, DaliCoreSocial can easily be bundled with the specific Web application. If the application is a Java client application, a back-end component is still necessary, because most social media require a static callback URL to be available. Upon successful authentication, the social network wants to call a callback URL, and this URL should be under the control of the application. Fortunately, DaliCoreSocial takes care of this, so JavaFX developers can focus on their JavaFX client code.

We will now build a very simple application with a Web client and a JavaFX client, which allows end users to authenticate themselves via Twitter or Facebook. The application will subsequently show a list of friends on a Web page or in the JavaFX application.

LISTING 3

```
<context-param>
  <param-name>dalicore.social.facebook.consumerkey</param-name>
  <param-value>123460584819473</param-value>
</context-param>
<context-param>
  <param-name>dalicore.social.facebook.consumersecret
  </param-name>
  <param-value>1234bbcb1e33406cb1973e9322396e69</param-value>
</context-param>
<context-param>
  <param-name>dalicore.social.twitter.consumerkey</param-name>
  <param-value>abcdc9ww7iVWygvgDmxmw</param-value>
</context-param>
<context-param>
  <param-name>dalicore.social.twitter.consumersecret
  </param-name>
  <param-value>abcdKj2k77jfZgPpwR3eyd4Sgkrm3ZnFDLaOw7VU
  </param-value>
</context-param>
```

Download all listings in this issue as text

Application Logic

The demo application, called ConnectSimple, is a Maven project. The dependency on DaliCoreSocial in the pom.xml file of the project guarantees the availability of the REST endpoint required for logging in to a social network. The location of this handler is /dalicoresocial/connect{network}?callback={some_page}, where {network} can be twitter, facebook, or googleplus.

If a value for the callback parameter is specified, the user will be redirected to this page after successfully logging in. DaliCoreSocial will add to the callback function a query parameter that is a combination of a key (uid) and a value (the user ID of the logged-in user). In case the callback value is omitted, the user ID for the logged-in user will be returned.

When a user connects to a social network for the first time using the above-mentioned REST endpoint, an OnlineAccount will be created. In case the user is not yet known, a User instance will automatically be created as well. The

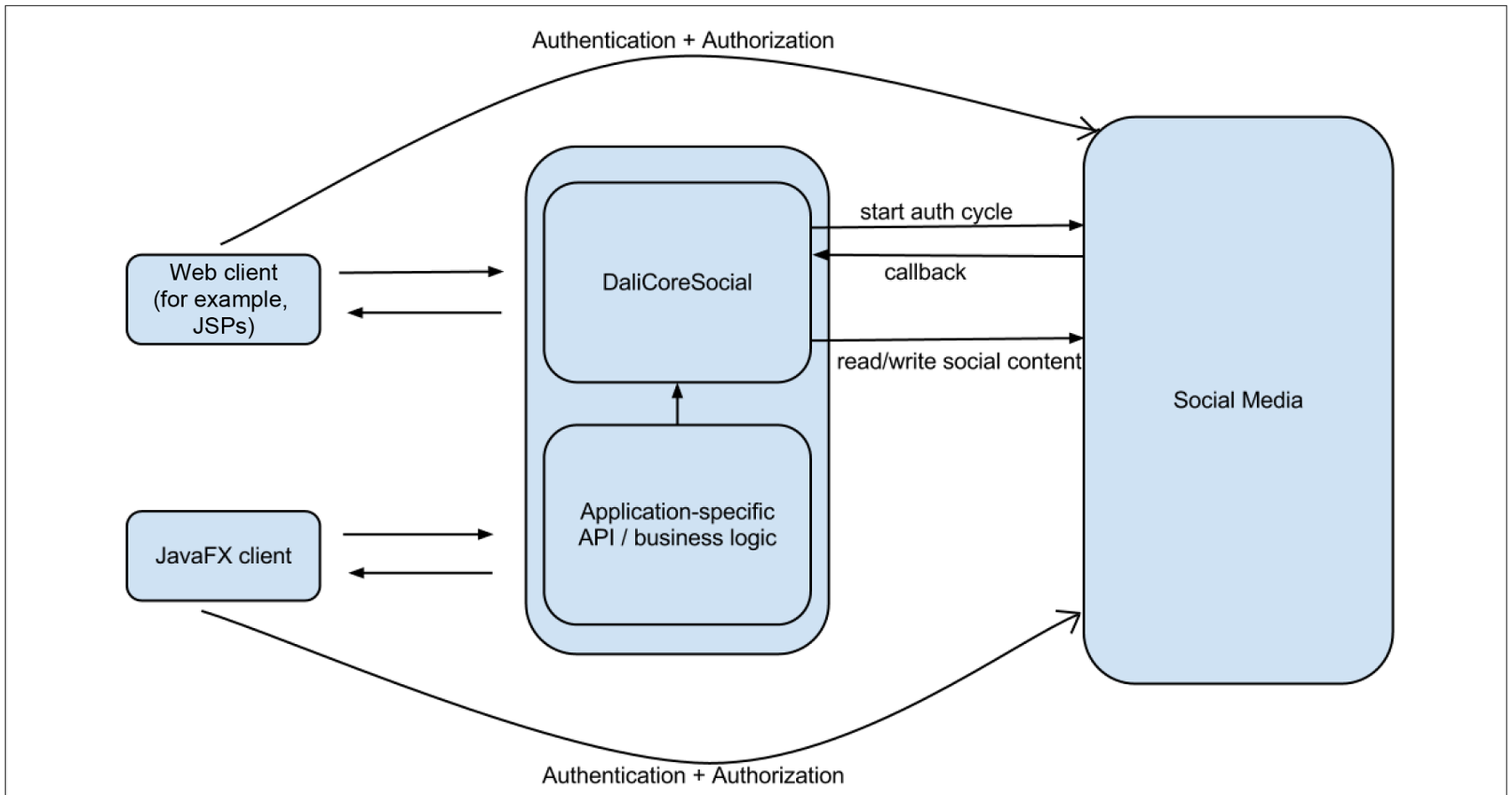


Figure 5

user corresponding to a specific user ID can be obtained by calling the following:

```
UserBean.getByUid(
    String uid);
```

The friends of a specific user can be retrieved by calling the following method, which will return the friends of the user for a specific network:

```
SocialBean.getFriends(
    String uid, String network);
```

Our demo application contains two REST handlers:

- Handler.java is used by the Web example. The responses of the method calls are Jersey Viewable instances, which create HTML pages.
- API.java is used by the JavaFX example. The responses of the method calls are in XML or JSON format and can be processed by any client capable of doing XML/JSON processing.

The functionality provided by both REST handlers is similar.

Web Example

The Web application contains a home.jsp file that contains three images: a Facebook image, a Twitter image, and a Google+ image. Clicking one of those images will direct the user to the start of the login flow at one of the following, which will then direct the user to the specific social network login page:

- BASE/rest/dalicoresocial/connect/facebook?callback=BASE/rest/demo/callback
- BASE/rest/dalicoresocial/connect/twitter?callback=BASE/rest/demo/callback
- BASE/rest/dalicoresocial/connect/googleplus?callback=BASE/rest/demo/callback

The whole authentication flow (including retrieval of a user access token) is done by DaliCoreSocial. When the user is authenticated and the social network allows our application to access the user information, an OnlineAccount is created and the user is redirected to BASE/rest/demo/callback.

Our application contains a REST handler class named Handler.java, which is annotated with the @Path annotation to indicate that it listens to REST requests that start with "demo":

```
@Path("/demo")
@ManagedBean
public class Handler {
...
}
```

Note that the class is also annotated with @ManagedBean in order to easily use the stateless session beans UserBean and SocialBean:

```
@Inject
SocialBean socialBean;
@Inject
UserBean userBean;
```

The callback method is anno-

tated with the @Path ("callback") annotation and will be called when the user is redirected to BASE/rest/demo/callback. This method is called by DaliCoreSocial after the social network redirects the user back to our application. DaliCoreSocial adds to the callback a query parameter, uid, which contains the unique identifier for the created user. The callback method will retrieve the user linked to the unique identifier by using the following code:

```
User user = userBean.getByUid(uid);
```

The OnlineAccount instances associated with the user are obtained by calling the code shown in **Listing 4**.

Both the user and the list of OnlineAccount instances are put in a Model class, which is a POJO that holds a user and the list of OnlineAccount instances. Next, a Jersey Viewable instance is created by calling new Viewable("/accounts.jsp", model); and this Viewable is returned. This will cause the accounts.jsp Web page to be displayed, and the information available in the Model instance can be rendered in the Web page, as shown in **Listing 5**.

The full code for this example is downloadable here and can also be viewed online.

JavaFX Example

We will now show how a JavaFX application can access information in social media, leveraging the same principles as in the case of a Web application. Our JavaFX example contains five classes:

- ExternalNetworkFX is the main class. This class, which also contains the functionality for communicating with the back end using REST, contains the current "state" of the application (for example, logged-in user and list of friends). In typical business applications, this functionality belongs to separate classes. For simplicity, we combined most of the controller and the model parts into a single class.
- ProfilePane is the class that describes the layout of the pane that is shown once the user has been successfully authenticated.
- UserDetail is the class describing the layout for a single user.
- User and Friend are two classes that hold data for the current user and the list of friends.

The typical flow in social network–based authentication leverages Web components. The application redirects the end user to the login page of a social network, and upon successful authentication, the social network redirects the end user back to the application.

LISTING 4 LISTING 5 / LISTING 6

```
List<OnlineAccount> onlineAccounts =
    socialBean.findOnlineAccountsByUser(user.getId());
```

Download all listings in this issue as text

JavaFX comes with a WebView component. We can use a WebView for the Web-based user authentication to a social network. Once the user is authenticated, we want to use instances of the plain JavaFX Node in order to render information, for example, a list of the user's friends. As a consequence, information about the logged-in user needs to be transferred from the WebView to the other JavaFX components. We will build an application that allows this. The code for the application can be downloaded here.

First, we add three image buttons that have the Facebook, Twitter, and Google+ icons. Clicking any of those buttons creates a WebView, and the location of the WebEngine is set to SERVER/dalicoresocial/connect/{network}, where network is either facebook, twitter, or googleplus, depending on which framework we want to connect. In our code, this is achieved for the Facebook button using the code in **Listing 6**.

For the Twitter button and the Google+ button, a similar snippet applies. The WebView will now point to the Facebook login page, as shown in **Figure 6**.

After the user grants the permissions, the social network will make a call to the DaliCoreSocial back end. DaliCoreSocial will continue with the OAuth flow, and a user access token will be obtained. DaliCoreSocial then checks whether the logged-in user already exists in the application and, if not, an OnlineAccount is created and coupled to a newly created User entity.

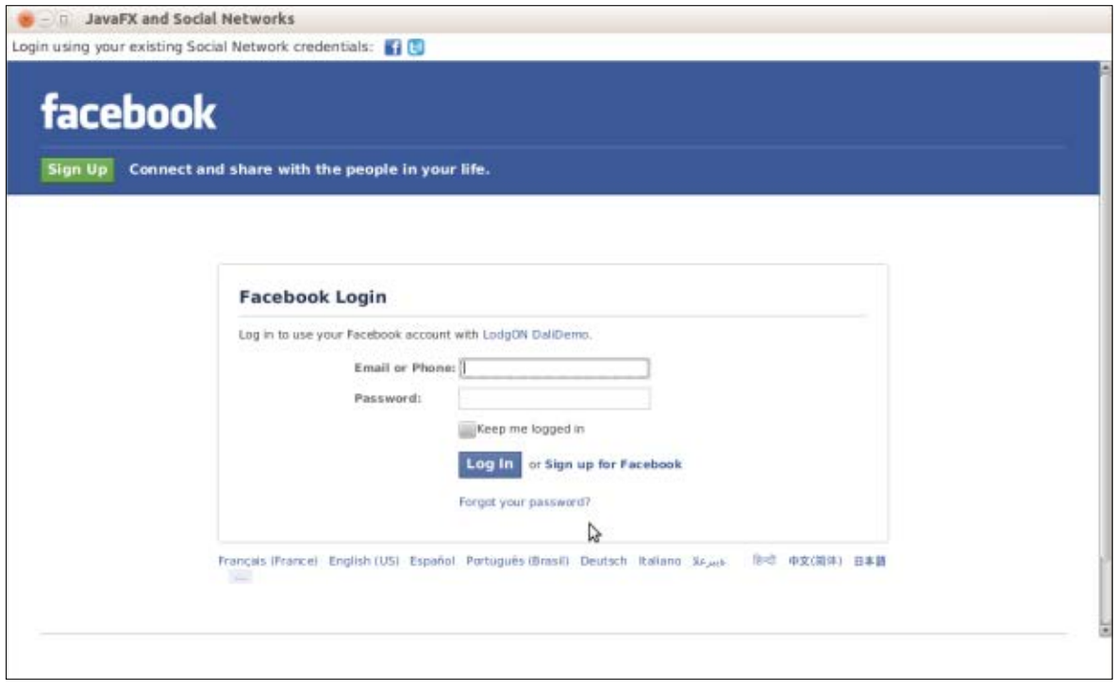


Figure 6

LISTING 7 LISTING 8 / LISTING 9

```
Node item = webEngine.getDocument().getChildNodes().item(0);
String uid = item.getTextContent();
```

Download all listings in this issue as text

Again, note that in case the authentication flow begins with a call to [SERVERBASE+"/connect/facebook/{uid}"], the created OnlineAccount will be coupled to the existing user with the corresponding uid, which is a UUID. Because no callback was specified in the initial call, DaliCoreSocial will respond to the client with the UUID of the logged-in user.

In the JavaFX application, we want to be notified when the location of the WebEngine starts with the callback URL of our application. After successful authentication, the social network will redirect our WebView to call the callback URL of DaliCoreSocial and, in our case, DaliCoreSocial will put the UUID of the user in the response object. We can read this UUID by parsing the Document Object Model (DOM) in the result page, as shown in **Listing 7**.

The obtained uid is an identifier for the logged-in user, and we can now start working with it. For example, the friends of a user can be obtained by making a call to the following REST endpoint:

```
GET /api/friends/{uid}
```

Note that the uid that has to be provided is the UUID of the user, and it is not the identifier of, for example, the Twitter or Facebook account. DaliCoreSocial keeps track of the OnlineAccount instances (for example, a Twitter or Facebook account) that are coupled to this user. As described earlier in the "Application Logic" section, the rest request will return all the friends that are on all this user's coupled networks. The code snippet in **Listing 8** shows how to obtain the friends.

EASY, OR NOT
The level of integration between an application and social media varies from easy to more challenging.

We used the DataFX framework for making a call to a REST endpoint and to parse the results. DataFX was described in a previous issue of *Java Magazine*.

The RestRequestBuilder will create a NetworkSource that makes a request to the back end and calls the method associated with the rest/api/friends/{uid} path. The ObjectDataSourceBuilder creates an ObjectDataSource that will parse the result of the REST call. We assume the information will be available in XML, and the different entities are tagged with the name "friend." The entities are converted to instances of the Friend class and added to the ObservableList<Friend> friends.

Visualizing the list of friends is done in the ProfilePane class. The code in **Listing 9** shows how we listen for changes in the list of friends. We loop over all the new

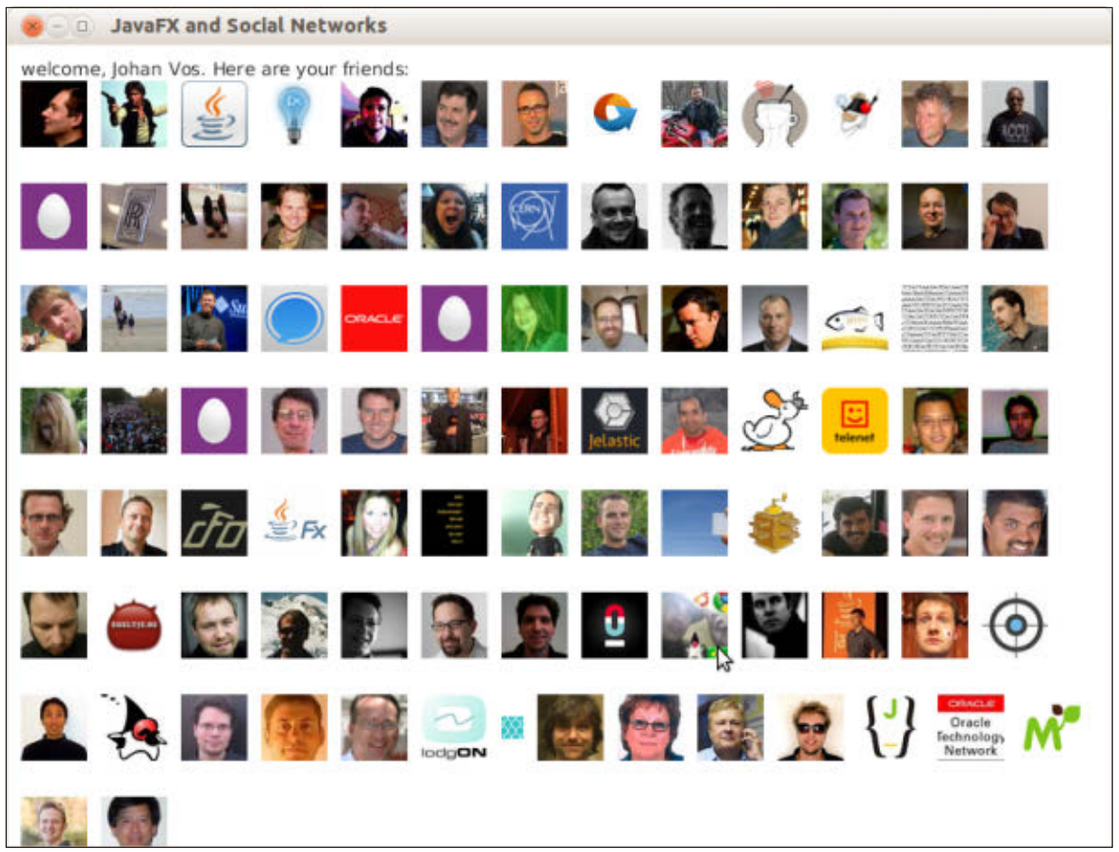


Figure 7

friends and create a UserDetail node for them. This node is added to a FlowPane. The result of this application is shown in **Figure 7**.

Conclusion

In this article, we showed how your application can integrate with large, existing social network systems. Authentication, privacy, and permissions are important concepts for all social media. The DaliCoreSocial module deals with these concepts, and provides Java developers with an easy-to-use framework that can be used to integrate social network users and functionality into your applications. It is important to note that the same approach works for Java EE applications (including Web clients) as well as for Java desktop applications or a combination of the two. `</article>`

LEARN MORE

- Johan Vos' blog
- Johan Vos' YouTube screencast on DaliCoreSocial

VIKRAM GOYAL

BIO

IMS Services API—Getting Started with JSR 281

Learn how to bring IP Multimedia Subsystem services to Java-enabled devices.

The IP Multimedia Subsystem (IMS) involves the evolution of very basic telephone and data exchanges into a modern exchange system, powered by a plethora of high-capacity devices and networking lines. Driven by the recent explosion in internet and high-end devices, Java ME, which is installed on billions of devices worldwide, is strategically placed to take advantage of this.

Here, I offer a brief introduction to IMS technology, followed by a discussion of JSR 281, the IMS Services API, which attempts to bring the IMS to Java-enabled devices. Finally, I will provide some simple code using the latest Java ME SDK for mobile phones (version 3.2) and the LG Session Initiation Protocol (SIP) server, which provides an implementation of this API.

Let's get started.

What Is the IMS?

The IMS is an architectural framework that helps with delivering multimedia services that use the Internet Protocol (IP). It was originally defined by the 3GPP (3rd Generation Partnership Project) as a standardized way for wireless mobile devices to distribute internet-based services. However, it is broader now and covers multiple network and media types.

The architecture defines a centralized exchange-type of system that advertises its services. End users can connect their devices to this centralized service and gain individual identities. Multimedia-based messages and data are sent to these identities via pipes that provide for simultaneous transfer of the data. Therefore, instead of an old-style telephone exchange system, which provides for only one kind of media transfer (voice), the IMS can easily handle multiple media transfers simultaneously.

This exchange of data can take place between services without a human element present, as in video or audio streaming where an end user consumes the data without needing to have a physical entity present at the other end. The end user is dialing into a service that would have advertised itself previously with its own unique identity. Business-to-business transactions are also possible, as long as individual identities have been established and a connection is possible based on pre-existing criteria and contracts.

Identity on an IMS network is established using SIP (RFC 3261—essentially, your e-mail address) or a "tel" URI (RFC 3966—essentially, your phone number). An end user or service may have multiple such identities, and the user can be reached on one, some, or all of these identities, *at the same time*. This is the beauty of an IMS-based system.

I have used the example of telephony so far, which would be the most common use. However, the structure of IMS precludes no particular data and is, therefore, media agnostic. You could easily use movies delivered over this architecture to illustrate its usage.

A commercial IMS system would control the functionality of the whole network

PHOTOGRAPH BY JONATHAN WOOD/ GETTY IMAGES

and provide the basic structure to get the system going. Thus, it would seamlessly integrate all available networks and provide authentication and authorization. Further, it could actually look up new networks and services and provide them to the end user for use. A recent example of this service is the Push-To-Talk feature that has been tried successfully by several telecommunications companies. Each IMS system should be able to advertise its services (or capabilities) when new subscribers connect to it.

If an IMS device provides new capabilities, these need to be separately identified. For each service that is standard, the 3GPP defines a standardized identifier called the IMS Communication Service Identifier (ICSI). For services that are new or localized to your application, the IMS Application Reference Identifier (IARI) is recommended.

One final point to note about the IMS is that it doesn't matter how a user connects to an IMS network; it always provides the same services to the end user. So a user could connect to a local IMS network via Wi-Fi at home, to a 3G or 4G network while traveling, and to a fixed-line network when at the office. At each place, the user experience within the IMS will be the same.

Java ME and the IMS

Java is present on a plethora of devices (not just mobile phones, but set-top boxes as well), so it is ideally suited to use the IMS. JSR 281, a step in this direction, defines an API that can be used to create an IMS-ready application that can sit on any Java-enabled device that implements this API.

A device that implements JSR 281 provides the *IMS engine*—an execution environment that an IMS-ready application can interface with. The application itself is defined with the following parameters:

- IMS application identifier (AppId), which is a string that identifies your application (potentially from among other IMS applications on the same device)
- The MIDlet, which handles the application logic and—in JSR 281 lingo—is said to be the "owning Java application"

The IMS engine stores these properties in the registry within the IMS device. This registry is used to identify the properties that define the capabilities of the IMS application.

Application developers need to define this registry by writing the capabilities within the Java Application Descriptor (JAD) or Java archive (JAR) manifest file. This is the static method of installing an IMS application on an IMS-capable device. JSR 281 also allows developers to dynamically register their application from within the MIDlet by using the classes in the javax.microedition.ims package.

The capabilities are defined in the registry by using at least one of the properties described in **Table 1**: StreamMedia, BasicMedia, FramedMedia, Event, or CoreService. The capability declara-

PROPERTY NAME	PROPERTY MEANING	EXAMPLE PROPERTY VALUES	SAMPLE DECLARATION
StreamMedia	MEANS THAT YOUR APPLICATION CAN STREAM AUDIO, VIDEO, OR BOTH.	Audio OR Video OR Audio Video	MicroEdition-IMS-1-Stream: Audio Video
BasicMedia	MEANS THAT YOUR APPLICATION CAN TRANSFER OR HANDLE MEDIA OF THE TYPES DEFINED BY THIS PROPERTY VALUE.	image/png, text/plain, application/myApp	MicroEdition-IMS-1-Basic: image/png
FramedMedia	SPECIFIES THAT THE APPLICATION CAN TRANSFER APPLICATION DATA-LIKE MESSAGES AND FILES. YOU NEED TO DEFINE THE CONTENT TYPES AND THE SIZE.	text/plain image/png, 4096	MicroEdition-IMS-1-Framed: text/plain image/png, 4096
Event	SPECIFIES THAT YOUR APPLICATION USES THE PUBLICATION AND SUBSCRIPTION CLASSES TO DECLARE EVENTS RELEVANT TO AN IMS APPLICATION.	presence	MicroEdition-IMS-1-Event
CoreService	DECLARES THE CORE SERVICES THAT THE APPLICATION SUPPORTS AND DEFINES ANY COMPOSITE PROPERTIES AND SERVICES THAT ARE SUPPORTED.	urn:URN-3gpp:org.3gpp.icsi;require;explicit FOR ICSI AND urn:IMSAPI:com.myCompany.iari.myApp FOR IARI	MicroEdition-IMS-1-CoreService-1

Table 1

tion must add at least one of these properties to identify the capabilities of the IMS application.

The API itself is divided into three packages.

As mentioned previously, the main javax.microedition.ims package is used to define, configure, and install an IMS application. The Configuration class provides methods such as setRegistry() and removeRegistry to dynamically install applications (as opposed to the static method just discussed). The ConnectionState class is used to monitor whether the application is connected to the IMS network and, if the network is connected, this class allows developers to find any identities on the network. Related to this class is the ConnectionStateListener interface, implementations of which can be used to retrieve notifications about changes in the state of the connection. Finally, this package also defines the Service interface, which is the base interface for all types of IMS services, including CoreService, which is an interface defined in the javax.microedition.ims.core package.

The second package, javax.microedition.ims.core, contains the interfaces and a single class that enable the application to create and provide the IMS services (in other words, the capabilities). At the heart of this is the CoreService interface, which provides methods for calling remote peers over the IMS network. The CoreServiceListener implementation of this interface is also used to listen for incoming connections. An example usage of this interface in shown in **Listing 1**.

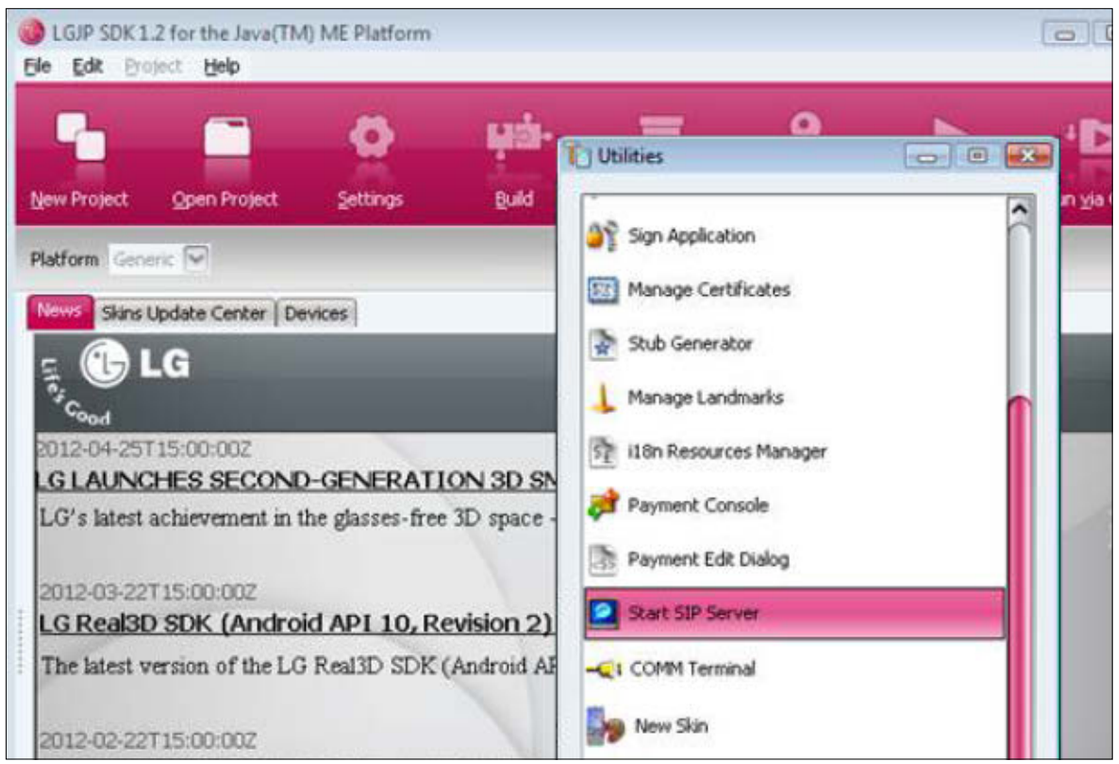


Figure 1

LISTING 1

```
service = (CoreService) Connector.open("imscore://myApp;
userId=Vikram sip:vikram@myemail.net serviceId=chat");
```

Download all listings in this issue as text

The CoreService is what enables connections and sessions within an IMS network, but it is the ServiceMethod interface and its implemented interfaces—Capabilities, PageMessage, Publication, Reference, Session, and Subscription—that are used to inspect and manipulate the messages that are transmitted within each session. The messages themselves are represented using the Message interface, but any manipulation of messages should be left for advanced IMS applications only. The MessageBodyPart interface allows an IMS application to create ad hoc body parts to attach to each message that is transmitted.

Each implemented interface of the ServiceMethod interface has a related listener interface attached. Thus, for the Capabilities interface, you can use the CapabilitiesListener; for PageMessage, you can use the PageMessageListener; and so on.

The final package, javax.microedition.ims.core.media, is used to help with the transmission of media (BasicMedia, FramedMedia, and so on). Consequently, it provides the BasicReliableMedia, BasicUnreliableMedia, FramedMedia, and StreamMedia interfaces. As the names suggest, BasicReliableMedia is used to transmit media over TCP, BasicUnreliableMedia is used to transmit media over UDP, FramedMedia is used to transmit data that is delivered in packets, and StreamMedia is used for media that can be streamed in real time.

A Simple IMS Application

In this section, I will walk through a simple IMS application for registering new users to an IMS server. More-detailed applications can be

built on top of this application.

Setting up. To get started, download the LG Java Mobile SDK, because we will use the emulation platform provided by it for the development of this MIDlet. It also provides a SIP server to test our MIDlet against.

Once you have the LG SDK downloaded, add it to NetBeans as another emulator platform. (NetBeans should auto-detect whether you are in the right section.) Make sure you select **IMS Services API 1.1** from the optional packages list.

Start the SIP server provided by the LG SDK by running the emulator environment. To do this, run the *installation folder*\bin\ktoolbar.exe file, and then select **File -> Utilities -> Start SIP Server**, as shown in **Figure 1**.

The LG SDK also provides the means to specify the location of the SIP server using the imsserver.ini file. This file is located in the *installation folder*\LGJP_SDK_12\appdb directory. Open the file and set the named variable com.lge.ims.ims_uri to the location of your SIP server (which is likely to be just the name of your machine; I tried using a local IP address, but that didn't work). You can leave the rest of the entries as is (at least for the purposes of this simple example).

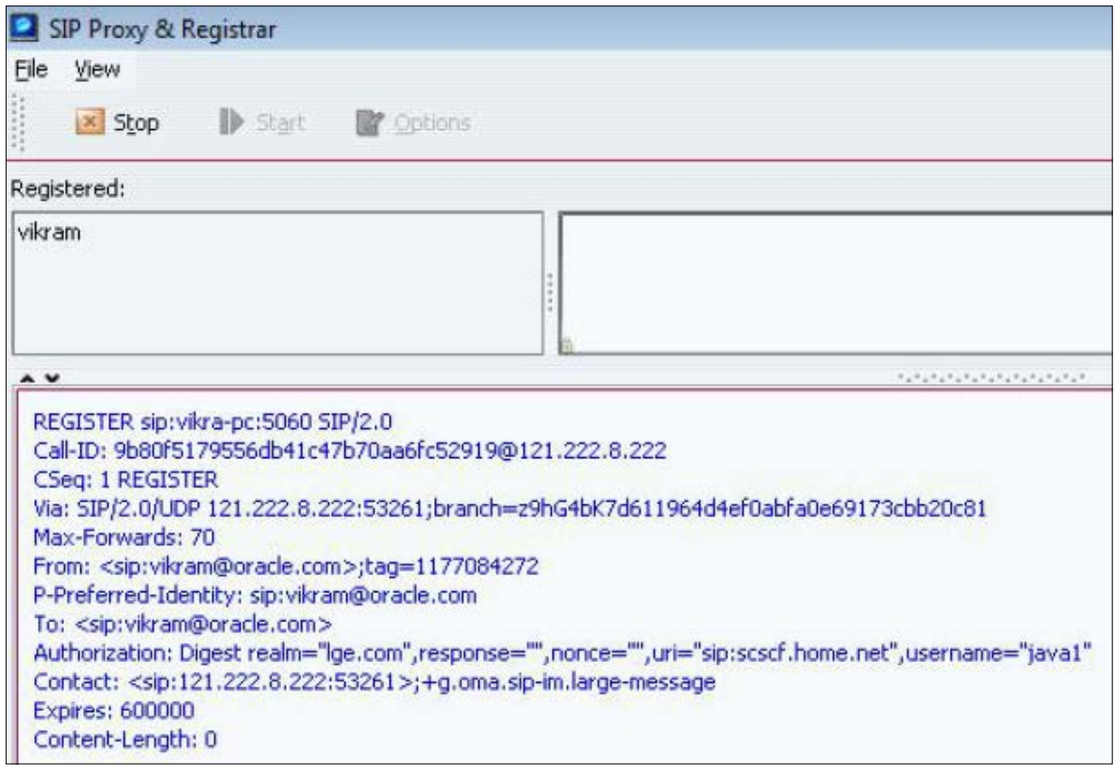


Figure 2

LISTING 2 LISTING 3

```
MicroEdition-IMS-1-Identity: com.oracle.test, IMSLaunchMIDlet
```

Download all listings in this issue as text

Next, there are three basic steps involved in creating a useful IMS application:

- Registering your application with your device
- Networking with an IMS server and declaring your identity
- Connecting a call and applying any media data, if applicable

Registering your application. To register your application, you can either use the JAD manifest way or the dynamic way (as I mentioned earlier). In this example, I have chosen to declare and register my application via the JAD manifest file.

The most important manifest entry that you need to provide is the MicroEdition-IMS-1-Identity property. This property should be set to the ID of your application and the fully qualified MIDlet name. See **Listing 2** for what I used for this application.

Networking with an IMS server and declaring your identity. In the testing world, networking with an IMS server is difficult to do because not many full-fledged IMS servers are publicly available. For testing purposes, I used the SIP server bundled with the LG SDK. There are some online SIP servers, but they are not reliable. (See this listing of public SIP servers.)

To start, use the code shown in **Listing 3** to declare your intentions to the IMS server.

Notice the connection string. It starts with imscore and then is followed by the predictable HTTP-like characters ://. Then, the next thing is actually the application ID that we declared in the previous section. Separated from this by a semicolon is the user who is making this request; this has a familiar e-mail syntax, preceded by sip.

The effect of this code is to register the user using the ID vikram@oracle.com with the IMS server defined in the previous section.

Figure 2 shows the response on the SIP server side.

Connecting a call and applying any media data, if applicable. Next, we

LISTING 4 LISTING 5

```
Session mySession =
  myCoreService.createSession(
    "sip:vikram@oracle.com", "sip:lisa@oracle.com");
mySession.setListener(this);
```

Download all listings in this issue as text

need to create a session to initiate a conversation or interaction. This is typically done by using the Session class, as shown in **Listing 4**.

This code assumes that the user on the other end (lisa@oracle.com) has accepted your session initiation request.

Once created, the session can be used to transfer any sort of data that the two end users and their devices are capable of, and all data can be transferred simultaneously. For example, the two users could initiate a chat application; have a text-based chat; and then during the same session, send data in the form of framed media (such as pictures or videos). See **Listing 5**.

Once the media type has been set, the session can be started. The actual creation of media is left for the developer of the application. For that, the developer can use the Mobile Media API (JSR 135).

Conclusion

The IP Multimedia Subsystem is a useful technology for leveraging internet-capable devices to deliver multimedia content seamlessly and reliably. Java ME takes full advantage of this by providing an implementation via JSR 281.

In this article, I introduced you to JSR 281 and how it fits within the whole IMS ecosystem. I started with an introduction to the IMS and then showed how JSR 281 implements the IMS architecture and enables developers to create applications based on this platform. Finally, I showed the steps required for creating a session within a MIDlet with this API. `</article>`

LEARN MORE

- IMS specification

Swing into Mobile

Use the Lightweight UI Toolkit on Nokia Series 40 phones.

NIRMAL KIZHAKKEVEETIL, **PRAKASH** RAMACHANDRAN, **REHA** CHAKROBORTY, AND **ANNA** ZHUANG

BIO

You've probably used Swing to create UIs for your desktop applications. The rich simplicity of Swing inspired the development of the Lightweight UI Toolkit (LWUIT), and this technology is now ready to use on the world's most popular mobile phone platform, Series 40.

With smartphone-like features at a budget price, Series 40 powers the latest Nokia Asha Touch phones, such as the Nokia Asha 311 and the Nokia Asha 308, shown in **Figure 1**. Series 40's capabilities are driving demand for Asha phones, and with that demand has come ever greater interest from users for smart apps, driving record downloads from the Nokia Store.

With an estimated 675 million Series 40 phones in active use every day, there is a huge potential market for your apps. With support for in-app purchase and in-app advertising, as well as integrated operator billing provided by the Nokia Store, the tools are there to monetize your apps.

In this article, you will discover how to take advantage of the demand for Series 40 apps by leveraging LWUIT through smart development tools and distribution through the Nokia Store.

LWUIT: Simplifying Mobile Apps

If you have flirted with mobile apps development, you might have been put off by not having a modern UI framework, needing to code everything yourself to the canvas, and having to build your own UI framework and controls for anything interesting or useful. "Why can't I just use Swing?" you might have wondered. You were not alone, and so LWUIT was born.

By providing a set of rich UI components, LWUIT simplifies many of the typical challenges you might face when creating apps for mobile phones. These components mean that with less effort, you can create apps that have compelling user experiences. Another advantage is that LWUIT comes with built-in support for touch- and keypad-based UIs, as well as multiple screen resolutions. With good design—by separating the UI from the app logic—you will be able to build a single UI that runs across multiple UI styles in different screen orientations and sizes and on devices with a range of form factors. And you can see this demonstrated in several LWUIT for Series 40 example applications on the Nokia Developer Website.

LWUIT Comes to Series 40

Nokia has a long tradition of supporting Java in its platforms and creating robust

Figure 1

and easy-to-use tools for developers. Adding LWUIT to the Series 40 platform was a natural extension of this tradition. The first goal was to ensure that LWUIT was available for all the UI styles found on Series 40 phones: the full-touch UI, the touch-and-type UI (a unique hybrid of touchscreen and ITU or QWERTY keyboards), and the nontouch UI.

Next, LWUIT for Series 40 had to offer users the same look and feel as the underlying UI. In Series 40, Nokia has already created a set of UI styles that provides users with a great user experience on phones that have relatively small screens. Therefore, Nokia optimized LWUIT so it better matched the platform UI styles. For example, the LWUIT text area and text field were tightly integrated with the native Series 40 text editor found in more-recent phones.

To benefit fully from the scalable UI components, LWUIT for Series 40 adds built-in support for dynamic orientation change in full-touch phones.

The touch UI is one of the key features of Nokia Asha mobile phones, and it provides support for a wide variety of gestures. Therefore, Nokia optimized and tightly integrated LWUIT for Series 40 with the native touch UI using a simple yet elegant solution: the GestureHandler helper class (part of the com.nokia.lwuit package), which can be used in your LWUIT-based apps, as shown in **Listing 1**.

Another key aspect of the LWUIT port to Series 40 is memory consumption and performance. Several key components—such as Form, List, and HTMLComponent—have been improved. In addition, UI widgets' usability has been improved for Series 40.

A SINGLE UI

By separating the UI from the app logic, you can build a single UI that runs across multiple UI styles in different screen orientations and sizes and on devices with a range of form factors.

Designing with LWUIT

The three UIs used in Series 40 phones have their own unique characteristics. While there is much in common among the different UI styles there are differences, most notably between the full-touch UI (shown in **Figure 2**) and the two UIs where a keyboard or keypad is present: the touch-and-type UI and the nontouch UI (shown in **Figure 3**).

LISTING 1

```
Form form = new Form();
    GestureHandler
handler = new GestureHandler() {
        public void gestureAction(GestureEvent e) {
          switch(e.getType()) {
            case GestureInteractiveZone
.GESTURE_TAP:
              //handle tap
              break;
            case GestureInteractiveZone.GESTURE_LONG_PRESS:
              //handle long press
              break;
            case GestureInteractiveZone.GESTURE_FLICK:
              //handle flick
              break;
          }
        }
      };
      //set handler to receive gesture events only when the
// form is visible. Every form can have its own handler.
      GestureHandler.setFormGestureHandler(form, handler);
      //Creating another handler
      GestureHandler g = new GestureHandler() {
        public void gestureAction(GestureEvent e) {
          //handle event
        }
      };
      //if you want to always catch gestures, set the handler
// as a global handler. There can be only one global handler.
      GestureHandler.setGlobalGestureHandler(g);
```

Download all listings in this issue as text

For example, all the Series 40 UIs employ a drill-down approach; back-stepping is hierarchical (not historical). By default, all the UIs have the back function in the lower right of the screen and in the Options menu, but the Options menu is repositioned in the full-touch UI. The primary action (called the *default command* in

LWUIT) is automatically mapped to the action button in the header bar of the full-touch UI and to the middle softkey in the other UIs. By default, the LWUIT for Series 40 application layout displays title and softkeys (or actions), but the application can be set explicitly to full-screen mode to provide more space for content.

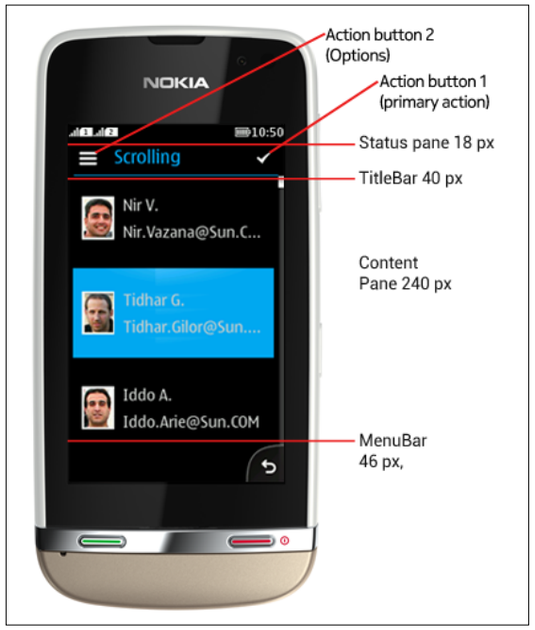


Figure 2

Figure 3

All the familiar LWUIT features are available, to enable the easy creation of impressive user interfaces. You can, for example, define application layout as a grid instead of the ordinary one-column layout. You can also use subtle animations (for single components) or transitions (for application views). Note, however, that excessive use of the effects is not encouraged, to ensure that application performance is good on all possible target phones. This is the area where developers may wish to consider feature variation based on phone capabilities; that is, build in more eye candy for more-capable phones such as the Nokia Asha 311.

One of LWUIT's unique benefits is the ability it provides for you to easily create custom themes using the LWUIT Resource Editor tool. Using the tool, it's possible to customize the LWUIT for Series 40 themes for a branded look and feel while maintaining a consistent UI.

The Designing for Series 40 section of the Nokia Developer Design and User Experience Library provides much more detail on this subject and is well worth visiting before you start design work.

Building Your LWUIT Apps for Series 40

Get the right tools. You build apps for the latest version of the Series 40 platform using the Nokia SDK 2.0 for Java. The SDK can be integrated with NetBeans or Eclipse, but the best experience will come from using the Nokia IDE for Java ME (Eclipse), which is installed by the SDK.

The Nokia IDE has a number of Nokia-specific features, such as a tool to help you locate the Series 40 SDKs you need and an editor for the Nokia-specific attributes in your apps' Java Application Descriptor (JAD) files.

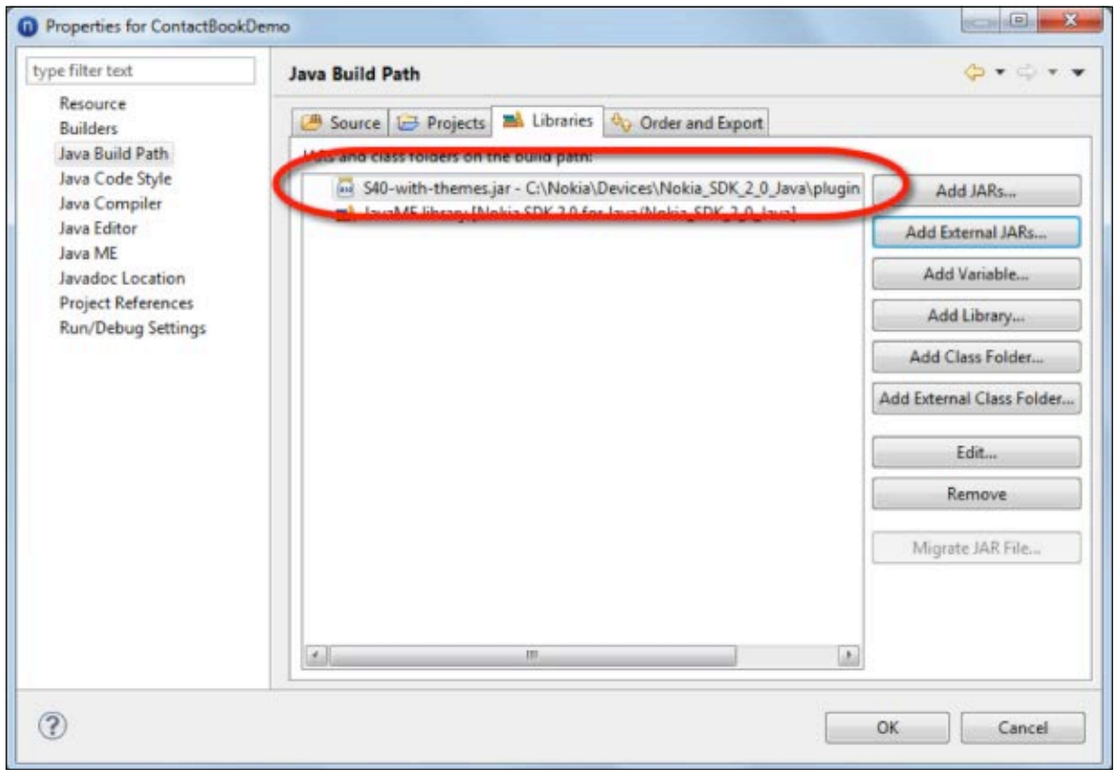


Figure 4

LWUIT is included in the 2.0 SDK. To use LWUIT in your apps, you simply add its Java archive (JAR) file to your app's build path and start adding LWUIT components, as shown in **Figures 4** and **5**. Beyond that, the development process will be pretty much as you would expect for a desktop app. The differences come when you start testing.

Test your app. To test your apps, you run them in a PC-based emulator. With the emulator, you are able to test virtually all aspects of your application, including location features or accelerometer sensors. You will want to test your

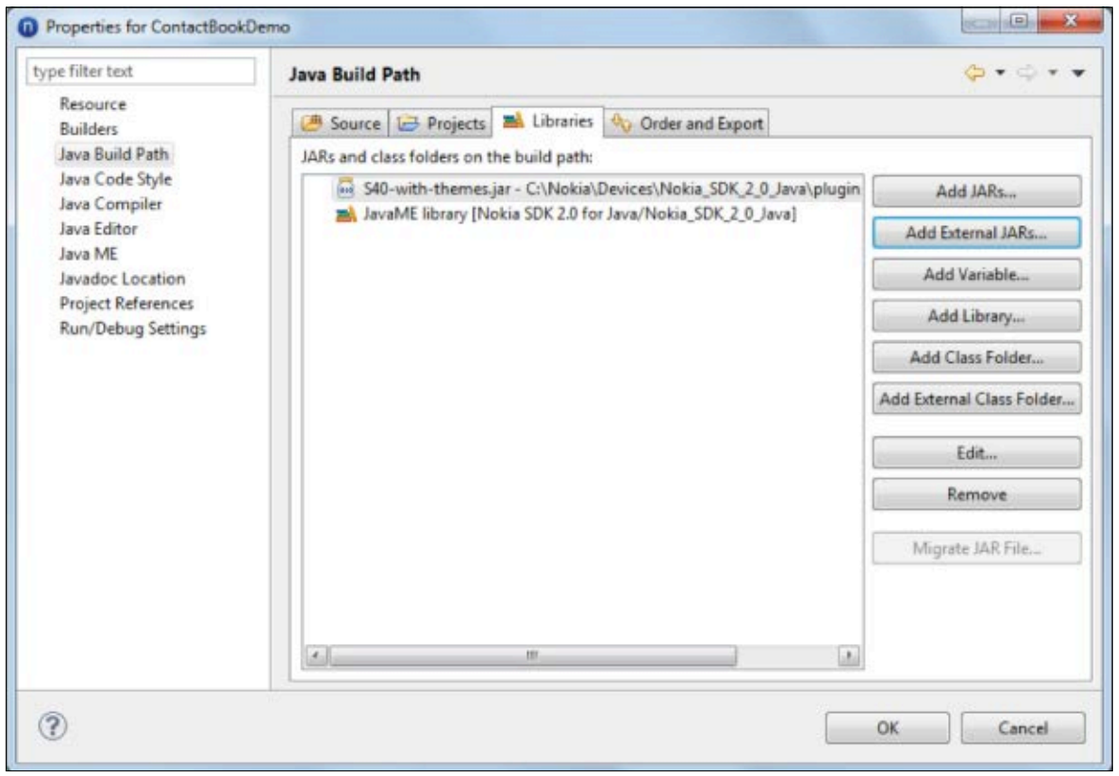


Figure 5

apps on a phone, too, but the emulator makes initial testing faster. **Figure 6** shows the Nokia SDK 2.0 for Java emulator running a LWUIT demo app.

And when it comes to testing on a phone, don't worry about getting access to Series 40 phones—you can access a range of Series 40 phones over the internet using the Nokia Developer Remote Device Access service.

Obfuscate your code. It is essential to obfuscate your code, because you include the LWUIT JAR in each application you build, and it adds up to 800 KB to each app's distribution file. By obfuscating, you can reduce this overhead to below 350 KB, depending on the complexity of the app (that is, how many LWUIT classes it uses).

Getting Started and Where to Get More Information

You can find more information on creating Java apps for Series 40 on the Nokia Developer Website. In addition to the SDK and design library, check out the videos and Webinars in the Learning section and the Java Developer's Library.

Getting started with LWUIT is just as easy. You can start from the LWUIT for Series 40 project home page, where you can learn and download the tools. Another recommended resource is the video introduction *Get Started with LWUIT for Series 40*, which you can watch on the YouTube channel (which includes many other videos about developing with Java for Series 40). In the LWUIT video, you will see how to set up the tools, code your app, test your app in the emulator, and get your app running on a Series 40 phone.

Figure 6

There is also a lively community forum in the Nokia Developer Java Tools discussion board, where you can post questions and join fellow developers in technical discussions.

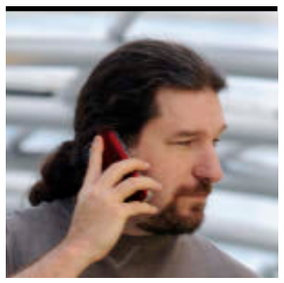
A Final Word

With the introduction of LWUIT, you will be able to create apps with a rich UI quickly. Then, with good design, apps can run unchanged over the range of Series 40 phones to cover the full-touch, touch-and-type, and nontouch UIs.

Because Series 40 phone users are beginning to demand more and more apps, now is a great time to start creating apps using the LWUIT library for Series 40 if you want to give your Java apps more reach. `</article>`

LEARN MORE

- LWUIT Javadoc
- Binary package
- "Working with Lightweight User Interface Toolkit (LWUIT) 1.4"
- "Rich Applications for Billions of Devices: What's New in LWUIT?"
- "Using Styles, Themes, and Painters with LWUIT"
- LWUIT for Series 40 example applications



TED NEWARD

BIO

Part 2

Building Actor-Based Systems Using the Akka Framework

Learn how to use the Akka library to do more than create simple actors.

In Part 1 of this two-part series, we discussed the difficulties in building reliable, distributed, concurrent systems, and we briefly explored the concept of an "actor" for building such systems. We explored an implementation of actors on the Java Virtual Machine (JVM) called Akka, which is part of the Typesafe Stack that also includes Scala and the Play framework. We illustrated some simple examples, but space constraints prohibited a deeper look.

I simplified the actor story a bit to help put the basics in place: The Akka documentation describes an actor as "a container for State, Behavior, a Mailbox, Children and a Supervisor Strategy." We've already seen the actor as a container for state, in that we saw an actor can hold state within it, and we've already seen the actor as a container for behavior, because actors contain code. That code, however, doesn't execute in response to method calls, but in response to messages sent to the actor via the actor's mailbox. This helps actors avoid blocking on one another, which is a major source of concurrency bugs.

The last two parts of an actor—children and supervisory strategies—become important when actors work together to accomplish some larger task, which is most of the time. The supervisory strategy will be a large part of how we follow Akka's ideas for building robust systems, which, if you recall, embraces a "Let it fail" concept rather than the "Keep it from failing, ever" strategy espoused by most traditional J2EE/Java EE stacks.

But let's take this one step at a time.

HUNG UP

An unbounded wait is essentially a deadlock or a production "hang" waiting to happen.

Minions! Where Are My Minions?

We've already seen a bit of how an actors-based system is designed in the earlier "Hello" and the Pi calculator examples, but these examples had one tiny flaw: If something went wrong inside the actor, the entire system (trivial though it was) would come crashing to the floor—and worse, never get up again.

In a trivial system that runs in the console, this isn't a problem—the user can always just rerun the command that launched the system. But in a system that's supposed to possess even a modest degree of reliability, this isn't so good.

Imagine, for a moment, that instead of simply printing to the console, our actors are engaging in some "risky" behavior, which in this case consists of doing anything that might throw an exception. We can mimic this by creating a case within HelloActor from our earlier examples that, in a semi-random fashion, throws an exception when used (see **Listing 1**).

This is obviously a contrived example, but it does create for us a failure scenario—one that allows for at least a certain amount of

PHOTOGRAPH BY PHIL SALTONSTALL

recovery—in this case, to just keep trying over and over again until we finally get messages out.

But we need to refactor the HelloActor design a little: HelloActor can't do everything on its own, because if the exception is thrown so that it escapes HelloActor, the actor dies. What we need is to establish a new actor that will do the "risky" operation, and then use that actor from within HelloActor, as shown in **Listing 2**.

When the code in **Listing 2** is run, an interesting thing happens: If the time happens to be lucky (or, I suppose, in this case, unlucky) enough to end in a zero, the exception is triggered, but the program doesn't terminate. Since the exception is thrown from a *child actor* (one that was created from within HelloActor), the actor's supervisory strategy—which is a OneForOneStrategy by default—looks at the exception thrown from within the child actor. Depending on the strategy's directive (a mapping of exception types to a directive type), the supervisory strategy decides which of the following to do:

- Resume the failed actor's message processing (meaning the actor didn't suffer a catastrophic failure and we can keep using it).
- Restart the failed actor (meaning the actor suffered a catastrophic failure and we need to create a new one to do message processing).
- Stop this actor.
- Escalate the issue to this actor's own supervisor (which, in the case of HelloActor, will be the Akka system runtime) by rethrowing the exception to it.

For example, by default, a RuntimeException maps to the restart option, which tells Akka to restart the failed actor. We know this by examining the type of the exception in the pattern match inside of the actor (see **Listing 3**).

In cases where we want entirely new behavior, we need to override the supervisorStrategy value. The canonical way to do this is within the actor subtype itself, as shown in **Listing 4**.

There's more to the SupervisorStrategy constructor, but the parameters maxNrOfRetries and withinTimeRange are fairly self-explanatory, are described in the Scaladocs for Akka, and have some generally useful defaults.

Context Matters

Notice that the code in **Listing 4** uses the *context value*, which is a bound value built in to every actor that provides a reference to the actor system in which it executes. This is a reference to an ActorContext object that has a number of methods that provide interesting information and behavior for an actor to use:

- actorOf: This method is used in **Listing 4** to create another actor that will be a child of this actor (the one doing the actorOf call).
- children: This returns an Iterable of references to all the child actors of this given actor.
- parent: This is the parent of this actor.
- self: This is a reference to the actor itself—this is a little different from the this reference, because the self reference is to the ActorRef that wraps the actor, rather than to the raw

LISTING 1 LISTING 2 / LISTING 3 / LISTING 4

```
class HelloActor extends Actor {
 def messageOut(msg: String) = {
  if ((System.currentTimeMillis() % 10) == 0)
   throw new RuntimeException("NOOO! Even time! The horror!")
  else
   System.out.println(msg)
 }

 def receive = {
  case Start =>
    messageOut("Hello, Akka")
  case incoming : RepeatMessage =>
    for (it <- 0 to incoming.num)
      messageOut("Repeat: " + incoming.message)
  case incoming : Message =>
    messageOut("Message: " + incoming.message)
  case Stop =>
    context.system.shutdown()
 }
}
```

Download all listings in this issue as text

object, and in most cases, self is the more appropriate of the two to use.

- stop: This method shuts this actor down.
- dispatcher: This returns the MessageDispatcher used for this actor. The MessageDispatcher is the strategy by which messages are delivered to the actor and largely describes the relationship of actors to threads. (For example, does each actor have its own thread? Or do all the actors operate on one thread?)
- system: This is the reference to the ActorSystem to which the actor belongs.
- sender: This is the actor from which the currently processing message came.

Almost any nontrivial actor will use context at some point during its execution, as we see in **Listing 4**. There are a few other methods on ActorContext, two of which we will examine later, but for the most part, these are well documented in the Scaladocs for Akka.

Don't Just Stand There! Say Something!

In Part 1, we showed that actors can send messages to another actor using the ! method, but nothing was said about how actors could engage in a round-trip request/reply. This was a deliberate omission on my part: The system as a whole functions a lot more quickly and smoothly if we design it so that all message-sends are considered one-way, asynchronous communications. No blocking means no opportunities for deadlock.

It's important to realize that an actor can always send a one-way message back to the sender of the message by using the sender reference within the actor. So it's always possible to do something like you can see in **Listings 5a** and **5b**.

But the problem with this approach is that if we fire messages at a RobinHoodActor using the traditional ! syntax, as shown in **Listing 6**, what we're going to get (see **Listing 7**) is not quite what we'd expect or, most likely, want.

Because the messages are asynchronous, Robin's messages to his men appear to arrive in an out-of-order fashion, and certainly the results returned to Robin are not quite in the order a leader usually expects (a response as soon as an order is given and before the next order is given).

In this case, it's possible to write request/response messages, but that approach intentionally requires a lot more work because, as already mentioned, blocking is playing with fire and should, by definition, require more thought and work. Even then, Akka won't give us an unbounded blocking request/response method. Instead, it will asynchronously send the message and hand back a Future object, which will (at some point) receive the result.

To start with, sending a request/response message uses the ask method—or its shorthand equivalent, ?—but a couple of things have to be established first, mostly dealing with how long the system should wait for a response.

To begin with, assume that we have a simple echoing actor, as shown in **Listing 8**. It's annoying, but it works. In the sender, once the actor is created, the message is sent using the ? method, which will work only if an implicit Timeout object has been created beforehand. (Otherwise, there will be a compile error. The ask

LISTING 5a | LISTING 5b | LISTING 6 | LISTING 7 | LISTING 8

```
case class RichMan(val kind : String)
case object Sheriff
case class Rob(val whom : String)
case class Hide(val where : String)
case class JobComplete(val msg : String)

class RobinHoodActor extends Actor {
 val rightHandMan = context.actorOf(Props[LittleJohnActor])
 def receive = {
  case rm : RichMan => {
   println(
   "Robin Hood says, 'Ho, ho! Rob the " + rm.kind + "!'")
   rightHandMan ! Rob(rm.kind)
  }
  case Sheriff => {
   println("Robin Hood says, 'The Sheriff! Hide, men!")
   rightHandMan ! Hide("forest")
  }
  case j : JobComplete =>
   println("Robin Hood says, 'My men report '" + j.msg + "'")
 }
}
```

Download all listings in this issue as text

method, on the other hand, is less syntactically short but takes the timeout parameter explicitly, which is sometimes a little easier to read.)

Once that's done, the returned object from the ? call is mapped to a particular future-result type. (In this case, because we're expecting a string back, it needs to be mapped back to a Future[String] type.) Then, finally, as shown in **Listing 9**, we need to establish how long we're willing to wait for that Future object to be populated with the result before the Akka API will allow us to actually harvest that result.

Why all the fuss? Michael Nygard, in his book *Release It!*, describes in great detail why unbounded waits are a bane to production-quality code. So I won't get into the details here, but in summation, an unbounded wait is essentially a deadlock or a production "hang" waiting to happen. The repeated use of timeout values here tells Akka exactly how long to wait for each potentially hanging operation before recovering and continuing execution (giving you the chance to recover from the lost message or response).

Note, though, that certain import statements (see **Listing 10**) must appear prior to the code in **Listing 9** for the code to compile, because neither the ask method nor the ? method appears on the Actor API. Failing to use these import statements will result in some truly misleading compiler messages.

Needless to say, Akka is making it a lot easier to use one-way messages, and despite the additional work, building redundant and fault-tolerant systems fares better when doing exactly that.

Get This to Phil in Accounting!

Seeing these messages flying from one actor to another might lead some readers to wonder if this isn't a replacement for Java Message Service (JMS) or some other kind of messaging-oriented middleware system. On the surface, yes, there appear to be some similarities, particularly when routing messages (our next topic), but similarities end there on the surface, really. A messaging-oriented middleware system is generally about making the messages and their delivery a top-level concern as something the developer wants to see and deal with, usually for reasons of durability, persistence, and so on. In an actor-based system, the messages are almost an implementation detail; they are used solely to help break the blocking nature of a method call and (in the case of the mailboxes themselves) are not something the developer wants or needs to deal with.

LISTING 9 LISTING 10 / LISTING 11

```
val ma = system.actorOf(Props[MockingActor])
implicit val timeout = Timeout(5 seconds)
val future = (ma ? "Hello").mapTo[String]
val result = Await.result(future, 5 seconds)
println("Received result: " + result)
```

Download all listings in this issue as text

However, there are interesting parallels between the two kinds of systems. One parallel is that of creating a "workflow" or "pipeline" of actors that can work together to accomplish a common goal. It might sound "retro," but building large systems out of small pieces—for example, by using UNIX command-line tools such as grep, sed, awk, and more—is by far a more resilient and robust way to achieve long-term success than by building monolithic large "chunks" of code.

In some cases, these chunks of code will be distinct and separate actor types (a PrintInActor, a FibonacciActor, a FactorialActor, or—if we want to get out of the mathematics domain—a ProcessRequestActor, a TransferMoneyActor, a NotifyAccountActor, and so on). And for some of these, linear sequences of steps through each (for example, A to B to C to D) will work, in which case, we can simply pass the "next" actor in to the previous one. But for others, some of the steps will be processed in parallel or in some combination or hybrid thereof. This entire subject is called *routing* in Akka, and while there's no way we can exhaustively cover all the different possibilities, we can explore a few simple scenarios.

Consider creating an actor specifically for printing to the console, as shown in **Listing 11**. In this particular example, we want to have five different instances of the PrintInActor (which we'll be able to verify by seeing the "created!" message printed when each one is instantiated). More importantly, when we send 10 messages, we

want the messages to be routed to each actor in a round-robin fashion, as shown in **Listing 12**.

When we run the code, what we see is that, indeed, five actors are created, and each one receives two messages in order, as shown in **Listing 13**.

The key to this is in the creation of the roundRobinRouter, an actor that essentially "fronts" the collection of five PrintlnActors such that to the client (main), it appears that there's just one actor, while in fact, the work is being split across five different instances, as we can see from the output.

Akka offers several different routers, each with different semantics:

- The RoundRobinRouter will pass messages to each actor in a round-robin, ring-like fashion, as we've already seen.
- The RandomRouter will, as its name implies, hand messages to a randomly selected actor with no other consideration.
- The SmallestMailboxRouter will examine the mailboxes of all the actors under it, and hand the message to whichever one has the smallest mailbox (meaning the least work being processed). Note that any remote actors will be selected last, since Akka can't see the mailbox state for actors running in remote processes.
- The BroadcastRouter will hand the message to all the actors underneath it in a broadcast fashion.
- The ScatterGatherFirstCompletedRouter is a BroadcastRouter but with a hitch. It will dispatch the message to each actor, but it will accept only the results of the first actor to return; it will throw away the rest of the results. This is essentially an implementation of the "Scatter-Gather" pattern commonly seen in messaging systems.

WORTH THE WORK

Akka makes it a lot easier to use one-way messages, and despite the additional work, building redundant and fault-tolerant systems fares better when doing exactly that.

Although we didn't do it, it's possible to create routers that fluctuate in the number of actors by creating a DynamicResizer and passing it in as a "resizer" argument when creating a router. Another option is to create a custom router, for example, a RedundancyRouter that would be similar to the ScatterGatherFirstCompletedRouter, but instead of taking the first result, it would take the first two results and compare them to make sure they're both identical, as they should be. The Akka manual has an example of one, if you find the need.

LISTING 12 LISTING 13 / LISTING 14

```
object Main {
 def main(args: Array[String]): Unit = {
  val system = ActorSystem()

  val roundRobinRouter =
   system.actorOf(Props[PrintlnActor].
    withRouter(RoundRobinRouter(5)), "router")
  1 to 10 foreach {
   i => roundRobinRouter ! i
  }
 }
}
```

Download all listings in this issue as text

You Must Become . . . Something Else

Of course, one of the most common things that we do when building systems is managing objects' internal state as clients use them in a progressive fashion. Akka offers a powerful capability for doing this for actors by allowing an actor to change the code that responds to the incoming messages. This capability hides behind the innocuously named become method, and at its heart, it's a simple mechanism: Each time become is called, a different method becomes the method that is executed when a message is delivered to the actor for processing. Consider **Listing 14**.

We can send the code in **Listing 14** some messages (the exact type of message is irrelevant to our example here, so we'll just

borrow the Start message from Part 1):

```
val pp = system.actorOf(
  Props[PingPongActor])
pp ! Start
pp ! Start
pp ! Start
```

What emerges at the console is this:

```
It begins!
PING
PONG
```

If this leads you to conclude that we could build finite state machine (FSM) behavior explicitly into the actors, you're spot-on. As a matter of fact, Akka provides some even more explicit support for describing an FSM in an actor, and the Akka manual describes the FSM module in more detail. (The become functionality is great for lightweight state transitions, but for very complex state machines, a more rigorous description of the states and their transitions will be necessary, and this is the support provided by the FSM module.)

We Need to Shoot a Scene in a Place Far, Far Away

It might come as a surprise that, as of yet, there's been no mention of remote actors. The main reason for this is simple: The programmer's approach to remote actors is, essentially, exactly the same as that for local actors, which is simply sending messages. The only things that are different are establishing a configuration that allows for remote communication (which consists of the right entries in a custom text file format that Akka can find when running your code) and specifying how to find (or create) the actor on the remote system. Neither of these tasks substantively changes the nature of your interaction with the actor nor the design of the actor-based system as a whole.

Configuration-wise, Akka will need an akka.conf file containing the entries shown in **Listing 15**. The actor section says that instead of using local actors, we now want actors that can receive messages from across the network, and the remote section specifies what communication transport and various TCP/IP–level details (such as the host name and port) to use.

In order to keep the external requirements to a minimum, I'm going to embed the configuration in the code and read it directly using a ConfigFactory (which, by the way, is not technically part of Scala; the Scaladoc is on the Typesafe Website). This approach is obviously not recommended for production code. The actor in question is shown in **Listing 16**, which is, again, kept deliberately simple to allow us to focus on the important parts.

Next, we need to establish the server on the remote machine, as shown in **Listing 17**. (Again, this will be the same machine we're running on to keep things simple.)

LISTING 15 LISTING 16 / LISTING 17

```
akka {
 actor {
  provider = "akka.remote.RemoteActorRefProvider"
 }
 remote {
  transport = "akka.remote.netty.NettyRemoteTransport"
  netty {
   hostname = "127.0.0.1"
   port = 2552
  }
 }
}
```

Download all listings in this issue as text

Aside from the configuration passed in to the ActorSystem, the server code is fairly standard Akka code. It creates the actor using actorOf, except this time, we give it a name in order to be able to find this actor from the client. We then block on an incoming keystroke (using System.in) just to keep the process up and running. (Again, this is not really recommended for production code.)

The client, then, is also almost just as deceptively simple (see **Listing 18**). The only thing that's new is that the method actorFor is used instead of actorOf, and the parameter to it is a URI describing where to find the remote actor. The format of the URI is fairly self-describing, as most URIs are, and most of the information was contained in the server's configuration:

- The scheme (akka)
- The server's ActorSystem name (server)
- The server's host name (in this case, the raw IP address 127.0.0.1)
- The server's port (2552)
- The actor's path (/user/actorName, where actorName was the string passed in when the actor was created using actorOf on the server)

When building the code, note that the system will now depend on a few more .jar files, notably the akka-remote JAR and the external dependencies netty-3.5.3.Final.jar and protobuf-java-2.4.1.jar (from the akka-2.0.3 release), all of which are already found in the Akka distribution but will likely need to be referenced somewhere in your build path or the project settings of your integrated development environment.

Conclusion

Akka is not, by any stretch of the imagination, a trivial system to master. The concepts are straightforward—message passing in a shared-nothing environment—but making the transition to building actor-oriented systems can be tricky. Just keep an eye on sending one-way messages, avoid blocking, and remember to let it fail! `</article>`

LEARN MORE

- Akka Website
- "Java Champion Jonas Bonér Explains the Akka Platform"

LISTING 18

```
object Client {
 def main(args: Array[String]): Unit = {
  val config = """
akka {
 actor {
  provider = "akka.remote.RemoteActorRefProvider"
 }
 remote {
  transport = "akka.remote.netty.NettyRemoteTransport"
  netty {
   hostname = "127.0.0.1"
   port = 2553
  }
 }
}
"""

  val system = ActorSystem("client",
   ConfigFactory.parseString(config))

  val actor =
   system.actorFor(
   "akka://server@127.0.0.1:2552/user/actorName")
  println("We have an actor: " + actor)
  actor ! "HELLO!!"
  println("Message sent")
 }
}
```

Download all listings in this issue as text

//fix this /

In the November/December 2012 issue, Jason Hunter and Boris Shukhat gave us a code challenge around porting JDBC code to use the Java Persistence API (JPA). They gave us code from a program that was redesigned to use JPA.

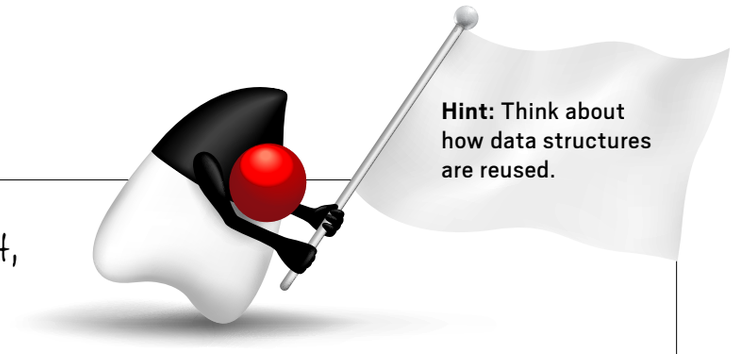
The correct answer is # 4. JPA commit() does a database rollback itself if it fails and then it closes the transaction. The correct code looks like this:

```
entityTransaction.begin();
try {entityManager.createNativeQuery(
sqlInsertStatement).executeUpdate();
       entityTransaction.commit();}
catch(RollbackException e) {log.info(
"Transaction failed and was rolled back");}
catch(RuntimeException e) {
if (entityTransaction.isActive()) {entityTransaction
.rollback();log.error(
"Transaction ended abnormally and was rolled back");}
}
```

This issue's code challenge comes from Simon Ritter, a Java evangelist at Oracle, who presents us with an embedded problem.

1 THE PROBLEM

Both the Connected Device Configuration (CDC) and the Connected Limited Device Configuration (CLDC) use the Connector framework to support generic input and output through a minimal-footprint API. CDC forms the basis of Oracle Java ME Embedded Client, and CLDC forms the basis of Oracle Java ME Embedded.



In the following example, we use a DatagramConnection to set up a server that listens and processes messages from clients. The messages are of variable length, but never more than 256 bytes. When the application runs, some messages require multiple reads to get the whole message.

2 THE CODE

How should this code be modified so all messages are read in a single getData() call?

```
datagram = connection.newDatagram(256);

while (notDone) {
  connection.receive(datagram);
  data = datagram.getData();
  bytesReceived = datagram.getLength();
  // process datagram ...
}
```

3 WHAT'S THE FIX?

1) Ensure that the type of the connection is UDPDatagramConnection.
2) Move the call to connection.newDatagram inside the while loop.
3) Add a call to datagram.reset() before the call to connection.receive().
4) Add a call to datagram.setLength(256) before the call to connection.receive().

GOT THE ANSWER?

Look for the answer in the next issue. Or submit your own code challenge!

PHOTOGRAPH BY BOB ADLER, ART BY I-HUA CHEN