RJS | PYTHON | SCGI | OPENOFFICE.ORG | Qt | PHP

LINUX JOURNAL

Since 1994: The Original Magaz

JUNE 2007 | ISSUE 158

METAPROGRAMMING PRIMER

VIEW CODE IN YOUR BROWSER

SCGI FOR FASTER WEB APPLICATIONS

OPENOFFICE.ORG EXTENSIONS

LUA OVERVIEW

Validate E-Mail Addresses in PHP

Interview with SUN'S CHIEF OPEN SOURCE OFFICER

The Object Relational Mapping Quagmire

www.linuxjournal.com

USA $5.00 | CAN $6.50

06

0 71486 03102 4

+ Qt 4.x Asynchronous Data Access

APPRO

HPC Cluster Solutions

Enterprise and High-Performance Computing Under Your Control

Appro is leading 4P x86 computing by combining the latest technology that meets the demands of the enterprise HPC market.

4-Way *Xtreme*Workstation™

- AMD Opteron™ 8000 Series processors
- Up to 128GB of DDR2 533/667 memory
- Up to 6.0TB SATA or 2.4TB SAS
- Hot-swappable drives
- 2 PCI-E x16 slots for high-end graphics card
- Windows® or Linux OS

4-Way 3U *Xtreme*Server™

- AMD Opteron™ 8000 Series processors
- Up to 128GB of DDR2 533/667 memory
- Up to 4.5TB SATA or 1.8TB SAS
- 2 PCI-E x16 and 3 PCI-X slots
- Hot-swappable drives
- Redundant power supplies & fans
- ServerDome Management – IPMI 2.0
- Windows® or Linux OS

For more information, please visit www.appro.com
or call Appro Sales at 800-927-5464

AMD Opteron™ Processors:

- Quad-Core Ready - increase capacity without altering datacenter infrastructure
- Best performance per-watt with energy-efficient DDR2
- Optimized system performance with Direct Connect Architecture

AMD 64 Opteron™

Today, Jack configured a switch in London, rebooted servers in Sydney, and watched his team score the winning run in Cincinnati.

With Avocent data center management solutions, the world can finally revolve around you. Avocent puts secure access and control right at your fingertips – from multi-platform servers to network routers, remote data centers to field offices. You can manage everything from a single screen, from virtually anywhere. This means you can troubleshoot, reboot or upgrade your data center devices – just as if you were sitting in front of them. Avocent simplifies your workday. What you do with the extra time is up to you.

For information on improving data center performance, visit

www.avocent.com/control

Avocent®
The Power of Being There®

CONTENTS

JUNE 2007
Issue 158

COVER STORY

FEATURES

ON THE COVER

The competition doesn't stand a chance.

If you base deployment decisions on performance and price, Coyote Point's for you. We've cornered that market.

To prove it we asked The Tolly Group to evaluate our E350si application traffic manager against the competition. The results speak for themselves.

Throughput? Almost 40% more than others in our space. Cost of transactions per second? Up to four times less. Connection rate? In some cases, one-sixth the cost. One-sixth! And we're told Coyote Point is the #1 choice for today's open source networks.

But don't just take our word for it. Get the facts. Call 1.877.367.2696 or write info@coyotepoint.com for your free copy of the full Tolly Report.

CoyotePoint
Systems Inc

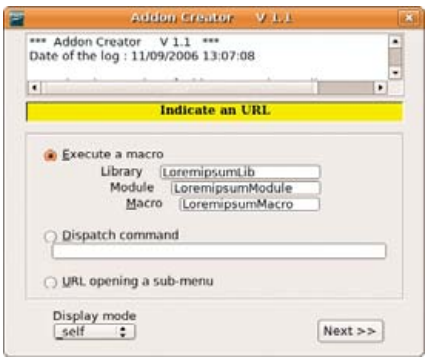
CONTENTS JUNE 2007 Issue 158

COLUMNS

IN EVERY ISSUE

QUICK TAKES

INDEPTH



Next Month

IMAGE PROCESSING

The third *Shrek* movie involves vastly more complex image processing compared to the first two. Next month, we explore the details of how DreamWorks Animation put Linux through 20 million CPU render hours to produce this amazing work of art. Want to switch to Linux but can't do it without Adobe Photoshop? Our interview with Pavel Kanzelberger, the creator of the Photoshop Linux work-alike, Pixel, is a must read for you.

As always, there's much more. For example, we give you details on how to implement Internationalization and Localization in your code, and we continue our tutorial on Python with the basics we bypassed in the first installment.

USPS *LINUX JOURNAL* (ISSN 1075-3583) is published monthly by Belltown Media, Inc., 2211 Norfolk, Ste 514, Houston, TX 77098 USA. Periodicals postage paid at Houston, Texas and at additional mailing offices. Cover price is $5 US. Subscription rate is $25/year in the United States, $32 in Canada and Mexico, $62 elsewhere. POSTMASTER: Please send address changes to *Linux Journal*, PO Box 980985, Houston, TX 77098. Subscriptions start with the next issue.

Gemini 2U

– Ultra Dense Series

- Two fully independent systems in a 2U
- Ability to run two discrete operating systems in one box
- Up to 16 CPU cores
- Up to 12 hot-swappable SATA, SCSI, or SAS hard drives
- RAID 0, 1, 5, 6, 10, 50 available on both systems
- Opteron™ or Xeon™ multi-core processors
- Up to 64GB memory per motherboard
- One available PCI-X or PCI-E slot per motherboard
- High efficiency AC and DC power options

2 Systems in One

Built on open standards, the Gemini 2U elegantly accommodates two discrete motherboards in a 25" chassis uniquely designed for easy access from the rear.

Gemini 2U represents the realization of intoxicating power and superior environmental specifications, with considerably less power consumption, less heat and less noise. Remarkably, it all fits nicely into any standard rack. Front to back, Gemini 2U is both powerful and efficient.

At Open Source Systems we understand you need practical, customizable, and affordable solutions that are easy to manage and maintain.

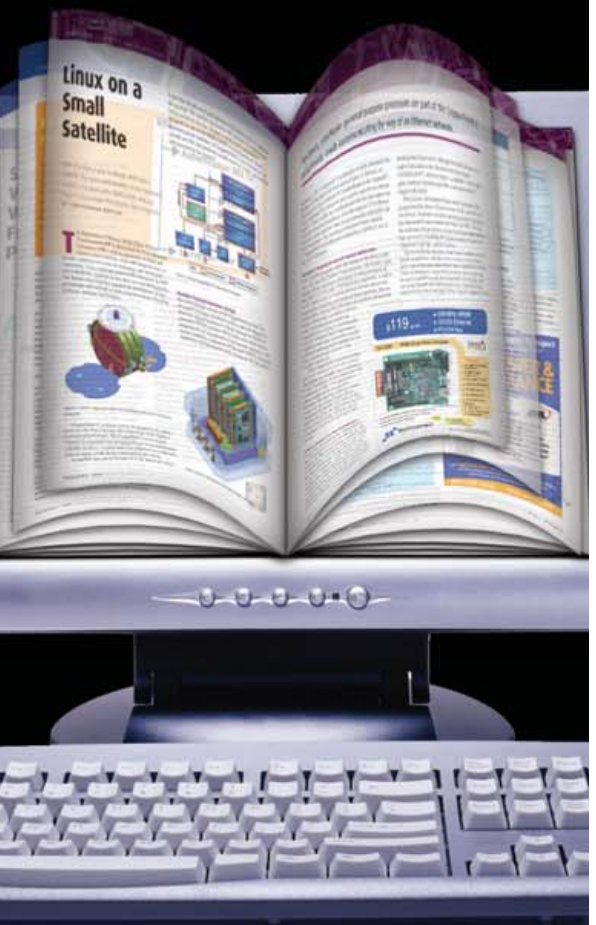
For more information and to request your evaluation unit today, visit us at **www.OpenSourceSystems.com**, or call direct at 866.664.7867.

Patent Pending

AMD Solution Provider PLATINUM

AMD 64 Opteron™

Open Source Systems™



LINUX JOURNAL

Editor in Chief
Nick Petreley, ljeditor@linuxjournal.com

Executive Editor	Jill Franklin jill@linuxjournal.com
Senior Editor	Doc Searls doc@linuxjournal.com
Art Director	Garrick Antikajian garrick@linuxjournal.com
Products Editor	James Gray newproducts@linuxjournal.com
Editor Emeritus	Don Marti dmarti@linuxjournal.com
Technical Editor	Michael Baxter mab@cruzio.com
Senior Columnist	Reuven Lerner reuven@lerner.co.il
Chef Français	Marcel Gagné mggagne@salmar.com
Security Editor	Mick Bauer mick@visi.com

Contributing Editors
David A. Bandel • Greg Kroah-Hartman • Ibrahim Haddad • Robert Love • Zack Brown • Dave Phillips • Marco Fioretti • Ludovic Marcotte • Paul Barry • Paul McKenney • Dave Taylor

Proofreader Geri Gale

Publisher	Carlie Fairchild publisher@linuxjournal.com
General Manager	Rebecca Cassity rebecca@linuxjournal.com
Director of Sales	Laura Whiteman laura@linuxjournal.com
Regional Sales Manager	Joseph Krack joseph@linuxjournal.com
Regional Sales Manager	Kathleen Boyle kathleen@linuxjournal.com
Circulation Director	Mark Irgang mark@linuxjournal.com
Marketing Coordinator	Lana Newlander mktg@linuxjournal.com
System Administrator	Mitch Frazier sysadm@linuxjournal.com
Webmaster	Keith Daniels webmaster@linuxjournal.com
Accountant	Candy Beauchamp acct@linuxjournal.com

Linux Journal **is published by, and is a registered trade name of, Belltown Media, Inc.**
PO Box 980985, Houston, TX 77098 USA

Subscriptions
E-MAIL: subs@linuxjournal.com
URL: www.linuxjournal.com/subscribe
PHONE: +1 713-589-3503
FAX: +1 713-589-2677
TOLL-FREE: 1-888-66-LINUX
MAIL: PO Box 980985, Houston, TX 77098 USA
Please allow 4–6 weeks for processing address changes and orders
PRINTED IN USA

TYANPSC™

Hands On Supercomputing

Typhoon™ 600 Series Personal Supercomputer

TyanPSC's Typhoon™, the next generation turnkey Personal Supercomputer has the power to blow away all your computational needs! Purpose-built for office and laboratory environments, easy to deploy and use, the Typhoon™ provides intense computational power in remote or constrained places, works like a PC and is whisper quiet.

High Performance Computing Just Got Cooler

Clusters of Typhoons / Low Power
Small size Form Factor / Under Mobility

Typhoon

T-630 DX / T-650 QX Series

- Up to 186 / 256 GFlops at your desk!
- Turnkey, Easy-to-Deploy, and Easy-to-Use
- Integrated 5 node cluster - up to 20 / 40 processor cores in a box!
- Plugs into standard wall outlet - only uses 15 Amps
- Microsoft® Windows® Compute Cluster Server 2003 pre-installed
- High Performance in Constrained spaces: Office, Remote, Plane, Boat, etc
- RAID capable

intel Xeon inside™

Microsoft® Windows®™ Windows Compute Cluster Server 2003

TYANPSC™ Personal Supercomputer

Tyan Computer USA
3288 Laurelview Court
Fremont, CA 94538 USA
Tel: +1-510-651-8868 Fax: +1-510-651-7688
Pre-Sales Tel: +1-510-651-8868 x5120
Email: marketing@tyan.com

For More Information, please visit
www.tyanpsc.com

letters

In Praise of Linux Games

I'm increasingly impressed by the variety and quality of games that are available for Linux. Most of them are pretty wholesome, which I appreciate, as I don't care for on-screen violence any more than the real thing. Have you considered adding a column devoted to Linux games? I'd bet lots of people would read it. By the way, I've enjoyed Marcel Gagné's articles immensely and have tried a lot of the programs he has mentioned. Reading *LJ* is a great way to learn about interesting software. Keep up the good work!

--

Mike Ford

Now You're Cooking with Linux

I would like to address the critics of Mr Gagné. Marcel's column provides people who are new to Linux topics that are not way over their heads or beyond their needs. The different writing style/format makes the column even more appealing to the Linux newbie.

By attracting new desktop users, Mr Gagné makes a huge contribution to the Linux community. Mr Gagné does this both as a professional writer and as a citizen. Evidence of this can be found in Mr Gagné's WFTL-LUG. I would suggest that the complainers join the WFTL-LUG and find out for themselves. Perhaps they can make a contribution or two in the LUG instead of all this unproductive negativism.

By the way, the March 2007 edition was super. Keep up the good work.

--

John Kerr

Now You Don't See Them, Now You Do

In the April 2007 issue, Chris Trayner states in his letter "Now You See Them, Now You Don't" that Konqueror no longer allows one to right-click on a file and see an option to delete it. True, it is not there by default. However, if you go to Settings→Configure Konqueror→Behavior, you will see an option Show Delete context menu entries, which bypasses the trash can. Check this box, and the delete option will be present at right-click. Thank you for an outstanding publication!

--

Dwight Middlebrook

Thanks for clearing up that issue for us!—Ed.

Doc Searls, iPhone Home

In the April 2007 issue on page 42, "Why an iPhone When We Can Make Our Own OpenPhone?" by Doc Searls, the context that ANA is relating directly with customers via the cell is misleading. I have worked with all three carriers in Japan: NTT DoCoMo, AU and Yahoo Keitai (formerly Vodaphone Japan). Yes, there is an IC system that allows passengers to check in quickly. Yes, ANA will get stats on how many people are using the system, but JAL also has this system and also the Japan railways (JR East/West). The technology is based on Sony's contactless IC chip. But, the system is controlled by each cell carrier. The carriers also control the Java applet that runs the IC chip, so they too can receive stats on who is using the system. The carriers here in Japan control which phones their service will use. It's not as bad as in the US, as there are only three silos here. But, the customers have no rights. With locked SIM chips, you can't get *any* unlocking code. Until last year, there was no number portability. Just in the last five years, Japan cell phones could be used overseas without the blessing of the carrier. Interactive programs with the end user aren't possible. One additional comment, the only reason IC-enabled cell phones exist is because the train system started using IC-enabled train passes—JR being the majority of railways and majority of the population rides those trains every day.

--

Robert Balfour

Cell-Phone Silos

Doc, you didn't mention the worst cell-phone silo of them all [*LJ*, April 2007, "Why an iPhone When We Can Make Our Own OpenPhone?"]: the crippling system that makes it impossible for cell phones to communicate directly with each other. Whatever happens in software development for mobile phones, this silo will stand as an insurmountable obstacle to progress until the network model changes from the current client/server system to an unrestricted, peer-to-peer, ad hoc model.

This transformation is certainly disruptive technology. As such, it will probably follow the usual disruptive technology development path. It will be ignored, ridiculed, lobbied against and actively resisted by the existing technology providers, until it makes them irrelevant and takes over the world.

What if this technology had been in use when Katrina hit New Orleans? There would have been no communications problems into any area where there was at least one phone within range of another, at any time, including the height of the storm. Rescuers would not have had to wait for days or weeks for destroyed infrastructure to be replaced just to have basic communications.

Instead, there were thousands of working cell phones in the disaster area, all rendered completely useless by the silo owners' communication restrictions.

The future of wireless is already written large in the form of the Internet. A self-organizing, self-healing worldwide network is an unstoppable force. The next obvious step is to remove the wires, and in keeping with the best Free Software traditions, this can be accomplished from the bottom up, simply by doing it.

--

Carl Brown

Banning Novell or Buying SCO?

A few issues ago [March 2007, /var/opinion], Nicholas Petreley called for a ban on Novell after it signed an agreement with Microsoft. Today, I read on Slashdot that Novell assents to "Windows Is Cheaper Than Linux" (**news.zdnet.co.uk/software/0,1000000121,39286295,00.htm**).

My advice to Novell: buy SCO. They both seem to be on the same path.

--

Paul Ammann

News about Dell's Article

As a frequent reader of *LJ*, I must tell you about good news from Dell. For sure, in consequence of the article "A Modest GNU/Linux Proposal for Michael Dell" (from

TotalView® Introduces The Most Powerful Command For The Multi-core Age.

TotalView is the proven debugging solution built specifically to address your unique challenges when developing multi-core, multi-threaded applications.

As part of a complete suite of proven multi-core debugging and performance tools that supports C, C++ and Fortran on Linux, UNIX and Mac OS X, TotalView 8.1 is the only solution you can count on for all of your multi-core debugging needs. Developed to help debug the world's most demanding applications, TotalView is extremely powerful, yet easy to use. Visually-driven, it provides enhanced graphical representations that enable users to quickly recognize problems and zoom in to identify root causes. TotalView does not require instrumentation, relinking, or rebuilding. TotalView has been proven to reduce debugging time by up to 80%.

Try it now, for free! Go to www.totalviewtech.com/command to power-up your 15-day trial version, or call 1-800-856-3766 for more information.

TOTALVIEW TECHNOLOGIES

Built for the Multi-core age

www.informationweek.com):

> Dell launched a Linux Web survey this week, moving it a bit closer to reintroducing the open-source operating system as a factory-installed option for home or office use.
>
> The survey, which was posted Tuesday and runs through March 23, asks a variety of questions, including which Dell system respondents would like to see with Linux, what kind of computing chores they would use the machine for, what type of software support they would like, and the Linux distribution they favor.
>
> In launching the survey, Matt Domsch, Linux software architect for Dell, said in the company's official blog that Dell has been moved to action by the more than 110,000 requests for Linux computers on the company's on-line customer sounding-board IdeaStorm.

So, for those of us who love to work outside the Windows world, this kind of movement comes at a very important moment and shows to the "monolitic, one-way" CEOs and "Masters of the universe" that intelligent life exists here! Keep up the great work there!

--
Eduardo

Blast from the Past

While cleaning out my desk, I found my Winter 1996 Linux Internet Archive set. Linux that ran on 4MB of RAM, MFM drives, EISA buses, Gravis Ultrasound cards—it brings a tear to one's eye.

Anyway, thanks for the informative articles. I am a longtime reader (someday I will get a subscription) and fan! Keep up the good work!.

--
John Harper

Re: the "Someone Else May Have to Decipher Your Code" Letter in April 2007 Issue

In his letter titled "Someone Else May Have to Decipher Your Code Someday", Michael C. Tiernan suggested the use of temp files instead of pipes for readability, instead of thinking that pipes can be as easy to read as code used in temporary files.

But, when using temporary files in a production environment, it should be done right, so the lines:

```
Tmp1=/tmp/tmp.1.$$
Tmp2=/tmp/tmp.2.$$
Tmp3=/tmp/tmp.3.$$
Tmp4=/tmp/tmp.4.$$
```

should be replaced by:

```
Tmp1=`mktemp`
Tmp2=`mktemp`
Tmp3=`mktemp`
Tmp4=`mktemp`
```

or:

```
Tmp1=`mktemp /tmp/tmp.1.XXXXXXXXXX`
Tmp2=`mktemp /tmp/tmp.2.XXXXXXXXXX`
Tmp3=`mktemp /tmp/tmp.3.XXXXXXXXXX`
Tmp4=`mktemp /tmp/tmp.4.XXXXXXXXXX`
```

for security reasons. First, mktemp ensures the filenames are unused, and the files generated do have the access rights set to ensure that only the owner can read the content.

--
Berthold Hollmann

Miniature OpenGL Development System

I just thought you might want to know about this project: the myOS—Miniature OpenGL development system. It is a minimalistic OpenGL-capable GNU/Linux-based system without X. It is a bare-bones Linux system, stripped down of everything but the core necessary files to compile and run OpenGL/C code. It has a simplified directory structure and cleaned up internal cross-referencing. It starts up with, and in total has, only a single script (**one.xthost.info/zelko/opengl.html**).

--
ZeAtShuttle

Did you know Microsoft owns the patent to OpenGL?—Ed.

Maximize Opteron™ Quad-Core Performance with UniServer

Find Full Lines of Innovative Opteron™ Server and Workstation Platforms

From **1 socket**, up to **32GB** to **4 sockets**, up to **128GB**

- 1U Server - 1 & 2 sockets with up to 64GB memories
- 2U Server - 2 & 4 sockets with up to 128GB memories
- 3U Server - 2 & 4 sockets with up to 128GB memories
- Workstations - 1, 2 & 4 sockets with up to 128GB memories

www.uniwide.com

1 - 877 - 520 - 0071

UNIWIDE Technologies

UNIWIDE is an official AMD Validated Server Program Partner

AVNET 1 - 888 - 300 - 8277

BELL MICRO 1 - 800 - 291 - 2070

SYNNEX CORPORATION 1 - 888 - 756 - 4888

AMD 64 Opteron

diff -u

WHAT'S NEW IN KERNEL DEVELOPMENT

Ingo Molnar has implemented a new language-like wrapper system called **Syslets** to manage system calls from user space. These little mini-programs can run system calls asynchronously, responding to their behavior as the user desires, without having to return out of kernel space. Using Syslet wrappers, Ingo has measured 33.9% speedups over cached synchronous I/O and 19.2% speedups over uncached synchronous I/O. Interest in Syslets among kernel hackers is fairly high, though **Linus Torvalds** feels the Syslet programming interface is too complicated and difficult for casual users to experiment with. Clearly some work remains before Syslets are ready to go into the main tree.

Intel has produced a **PRO/Wireless 3945ABG Network Connection** adapter driver. Unlike some other Intel drivers, this one doesn't depend on a proprietary dæmon; it is fully open source. It does, however, require a microcode upgrade. This improved licensing situation is apparently not due to any different feature of the hardware, but rather is due to improvements in the microcode itself. Intel's driver has received a good reaction among kernel folks, and it seems on the way to being included in the source tree.

The well-known and long-standing **\0 loophole**, allowing drivers to pretend to be GPLed when they aren't, is being closed. Several folks are working on this, most notably **Jan Engelhardt**, who recently posted a patch to fix it. There's a controversy surrounding this patch, because some folks feel that if the kernel places greater restrictions on non-GPLed drivers than on GPLed ones, this constitutes a license enforcement feature, which could violate the terms of the GNU General Public License. So long as the loophole exists within the code, the debate can stay dormant, because driver writers can bypass the controls. Once the controls actually start to work, the debate gains immediacy for companies such as **LinuxAnt**, who have made use of the loophole in the past.

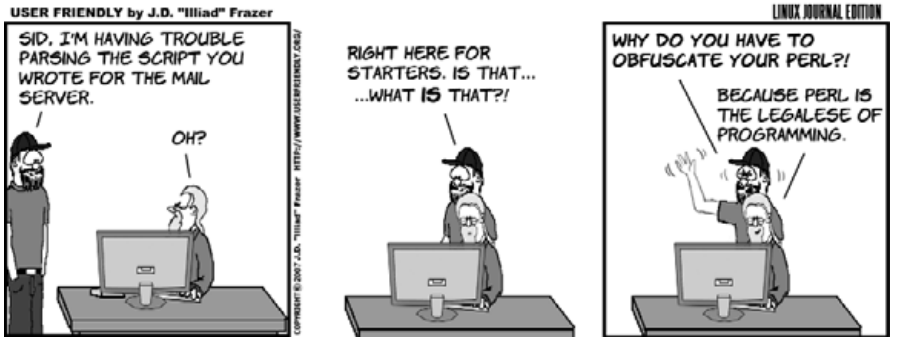
The **KVM virtual machine code** has migrated its development tree from Subversion to git for a couple reasons. **Avi Kivity** said, by way of explanation, that Subversion could not efficiently host the entire kernel tree and that developers wanted to maintain their own branches independently.

Jon Masters is the new **module-init-tools** maintainer, having taken over the job from **Rusty Russell** and patched the **MAINTAINERS file** to reflect the change. **Evgeniy Dushistov** also has created a **UFS entry** in the MAINTAINERS file, and listed himself as the maintainer.

Alessandro Di Marco got more than he bargained for when he posted a new **user inactivity trigger** he's been playing around with. It's a nice little feature that issues an ACPI event when no user activity occurs for a certain amount of time. He'd hammered the code out for fun, intentionally avoiding questions about optimal implementation details, on the assumption that not too many folks actually would be interested in this work. As it turned out, a lot of folks were interested, and they had many implementation suggestions. For starters, **Arjan van de Ven** pointed out that uevent would be a better delivery mechanism than ACPI. **Pavel Machek** pointed out that Alessandro's new /proc file would be better in the /sys directory. Pavel also suggested that user space would be a better place overall for such a feature, although Alessandro feels this would add a lot of complexity to the code. He responded quickly to a lot of the suggestions, producing new versions of his patch that answered the objections raised on the linux-kernel mailing list.

Planning for the next **Linux Kernel Summit** has begun. The kernel folks are spicing it up with a move to Cambridge, England, instead of the traditional Ottawa gathering in Canada. The new venue has opened up the possibility of different venues in the future, and a bunch of them have been proposed, including Australia, India and the Czech Republic. A major factor in selecting future locations will be the overall cost. A lot of kernel folks work at companies that pay for their plane tickets to the Summit each year, but some prices become prohibitive. Countries that are home to more attendees are more likely to host the Summit than others, according to **Theodore Y. Ts'o**, one of the organizers. But of course, anything can happen.

—ZACK BROWN



LJ Index, June 2007

1. Number of billionaires in the world: 946
2. Number of billionaires in India: 36
3. Indian billionaire gain over the prior year: 12
4. Position of India among countries with billionaires: 1
5. Number of billionaires in Japan: 24
6. Position of Bangalore among number of "Linux" queries on Google: 1
7. Position of Tokyo among number of "Linux" queries on Google: 2
8. Number of Western Hemisphere locations among the top ten sources of queries for "Linux" on Google: 0
9. Number of US locations among the top ten sources of queries for "Microsoft" on Google: 7
10. Millions of Internet radio listeners per month in 2005: 45
11. Millions of Internet radio listeners per month in 2006: 72
12. Dollars (US) paid per-listener/per-"performance" (recording) by US commercial Webcasters between 2002–2005: .0007
13. Same obligation as above for noncommercial Webcasters between 2002–2005: .0002
14. Dollars (US) to be paid per-listener/per-"performance" (recording) by all US Webcasters (commercial and noncommercial) retroactively for 2006: .0008
15. Dollars (US) paid per-listener/per-"performance" (recording) by all US Webcasters for 2007: .0011 per performance
16. Same rates for 2008: .0014 per performance
17. Same rates for 2009: .0018 per performance
18. Same rates for 2010: .0019 per performance
19. Position of Linux-based Radio Paradise as "most successful in its class" of Internet radio stations: 1
20. Royalty obligations of the above as a percentage of Radio Paradise's current total income: 125

Sources: 1–5: *Forbes*, CNN | 6–9: Google | 10, 11: *Radio and Internet Newsletter* | 12, 13: Librarian of Congress in 2002 | 14–18: Copyright Royalty Board in 2007 | 19, 20: KurtHanson.com

—Doc Searls

- **Root Access:** Providing the control you need.
- **Advanced FairShare Technology:** Better resource management means better performance.
- **Support That's Actually Supportive:** Award-winning support provided by system administrators.

Announcing Verio Linux® VPS.

At Verio, we have a long-standing commitment to open source, dating back to our roots in FreeBSD. Now, as the pioneer in virtual private server (VPS) technology and as a hosting provider backed by the financial resources of the world's largest telecommunications company, we bring something extra to Linux: reliability. To learn more, call 1-877-837-4654 or visit www.verio.com/linuxlineage.

There is no substitute for the right foundation.

Build on us.

VERIO

An NTT Communications Company

This Issue of *LJ* Dedicated to John Backus

I had written and submitted this month's Beachhead column, with a slant toward this issue's theme of computer languages, and in particular, the language FORTRAN, which was the first computer language I ever learned. And, although in my column I pointed out the benefit of learning machine and assembly language, I honestly do not believe that I would have gone into the programming field if it were not for FORTRAN.

On March 20, 2007, two weeks after I submitted the article, I got the very sad news of the March 17th death of John Backus. John was the man who people credit as the "Father of FORTRAN", and one of two people credited with Backus-Naur Form, a language invented to describe languages. I sent the message out to my local Linux user group, and during the next week it appeared again and again in various mailing lists.

With today's languages and computers, it is hard for people to know or (for those of us old enough) even to remember those early days. Today, concepts we take for granted were both revolutionary and difficult in conception in those days. There were people who thought computers would never be able to be programmed in anything other than machine code, and I am sure that John and his staff met more than their share of skeptics, but they persevered. And, out of the work they did on the first successful high-level language came many more successes by many more people on many more languages.

So I asked *Linux Journal* to dedicate this issue devoted to languages to John Backus: Computer Scientist Extraordinaire and Humanitarian, 1924–2007.

And, as we contribute our pieces to the future of computer science, may we hope someday in our own way to contribute as much as he did.

—JON "MADDOG" HALL

A Nice Handful

You've gotta like an MP3 player that goes out of its way to play OGG and brag about its Linux-friendliness (the literature says "Linux from kernel 2.4.x"). That would explain why TrekStor's Vibez player has been getting some nice buzz in Linux circles.

Upsides: you can load it as a plain USB storage device. It plays back OGG (plus MP3, WAV, FLAC and WMA). It has line-in and microphone recording, a color display, a USB-chargeable battery (plus a spare), device-deletion of files, adjustable play speed and a very slick non-iPod design. It's the size of a soap bar—4"x2"x.7", with highly rounded corners. And, it's a lot lighter—only 2.5 ounces.

Downside: not a lot of storage. Ranges from 8–15GB at prices that start at around $200 US.

—DOC SEARLS

IdeaStorm

Hardware OEM Learnings of Linux for Make Benefit Glorious Company of Dellstan

Early this year, Dell created a Digg-like site called Dell | IdeaStorm ("Where Your Ideas Reign"). The idea was for readers to submit ideas for the company, then have other readers vote and comment on them. Next to the title logo ran hints, such as "How can technology companies address climate change?" (over "click here to Read Dell's point of view"). Needless to say, this opened the floodgates holding back a tide of Linux demand.

As of 8am CDT on March 13, 2007, here were the top ten vote-getters:

1. 108,886 votes: "Pre-Installed Linux | Ubuntu | Fedora | OpenSUSE | Multi-Boot"
2. 73,840 votes: "Pre-Installed OpenOffice.org | alternative to MS Works & MS Office"
3. 54,300 votes: "Stripped down, fast Linux box"
4. 50,653 votes: "Have Firefox pre-installed as default browser"
5. 49,990 votes: "No OS preloaded"
6. 51,048 votes: "NO EXTRA SOFTWARE OPTION"
7. 35,867 votes: "Provide Linux drivers for all your hardware"
8. 29,041 votes: "Linux 2.6.16 ready (sticker)"
9. 20,288 votes: "National Call Centers"
10. 17,376 votes: "LinuxBIOS instead of proprietary BIOS"

One hour later, Dell added two more posts of its own. On the IdeaStorm page, it posted "Linux—We're listening—Now Tell Us More". On the Direct2Dell.com page, it added "Dell to Expand Linux Options". Both pointed to a survey at **www.dell.com/linuxsurvey**. The survey was also titled "Linux Learnings: We're Listening".

Hence the headline above.

—DOC SEARLS

Where Wants What?

One of the fun uses to which Google puts its vast Linux server farms can be found at trends.google.com. Here you can see and compare queries for keywords over a period of time that runs from the end of 2003 to the present.

In addition to showing trends on a graph, Google Trends also shows the top ten places from which queries come. This brings up some surprising results.

See if you can match the search terms on the left with the top search query locations on the right.

Answers on page 81.

—DOC SEARLS

1. gnome
2. kde
3. linux
4. shell
5. hat
6. hacker
7. laptop
8. widget
9. driver
10. emacs
11. vi
12. weenie
13. redhat
14. oracle
15. asterisk
16. internet
17. net
18. majordomo
19. maddog
20. vulnerability

A. Prague, Czech Republic
B. Pune, India
C. Oslo, Norway
D. Athens, Greece
E. Washington, DC
F. Honolulu, Hawaii
G. Tokyo, Japan
H. Rancho Santa Margarita, California
I. San Francisco, California
J. Stanford, California
K. Jakarta, Indonesia
L. Austin, Texas
M. Lima, Peru
N. London, United Kingdom
O. Thanh Pho Ho Chi Minh, Vietnam
P. Kyoto, Japan
Q. Bangalore, India
R. Bogotá, Colombia
S. Ljubljana, Slovenia
T. Hanoi, Vietnam

They Said It

The problem, of course, is that life is anti-formulaic, anti-institutional....Life can't be shrink-wrapped, caged, dissected, analyzed, or owned. Life is free.
—Christopher Locke, *The Cluetrain Manifesto*

I think a world full of anonymous monopolists is a really painful one to live in and create in.
—Mike Taht, the-edge.blogspot.com/2007/02/keeping-copyright-accessible.html

Good thing Henry Ford stopped in Waltham, Massachusetts, to learn about bicycle manufacturing rather than spending time in Mr Hobson's stable shoveling up after those horses.
—Bob Frankston, from an e-mail message

In isolation our wants exceed our powers. In society our powers exceed our wants.
—Frederic Bastiat, quoted in *The Logic of Co-operation* by George Jacob Holyoake, Co-operative Printing Society, 1873, www.citizenblog.org/node/23

The Industry Leader for Server Appliances

Custom server appliances or off the shelf reference platforms, built with your image and software starting under $1,000.

From design to deployment, we handle it all.

Delivering an appliance requires the right partner. MBX Systems is the right partner. We understand that by putting your name on our hardware, you're putting your reputation in our hands. We take that seriously. We provide the services you need to support your customers. Better than the competition. You never even need to touch the hardware. Engineering. Design. Deployment. We handle it all, so you can focus on what's important to you. Your software. Your sales. Your success.

Visit us at www.mbx.com or call 1-800-681-0016 today.

MBX systems

www.mbx.com | 1-800-681-0016

REUVEN M. LERNER

RJS Templates

The power of Ajax to fetch and run JavaScript generated by your server-side language.

Why not use your server-side programming language to generate the JavaScript for you?

The past few months, I've written a number of articles in this space about JavaScript. This language, built in to nearly every modern Web browser, has now come into its own and is at the heart of a paradigm for modern Web development known as Ajax. Whereas knowledge of JavaScript was long an optional skill for Web developers, it has become a must-have skill, along with SQL, HTML, HTTP and CSS.

One of the reasons for JavaScript's renaissance is the emergence of cross-platform libraries, which hide the incompatibilities that long plagued the language. For quite some time, programs written in JavaScript had to contain many if/then statements that looked at possible cross-platform incompatibilities.

Today, we can avoid having such if/then statements in our code by using libraries that take care of these low-level tasks for us. Prototype and Dojo, two of the JavaScript libraries I profiled in previous columns, have become popular precisely because they hide many of these details. They make JavaScript a truly cross-platform language, where "platform" means the Web browser as much as the operating system.

Some clever programmers, in an effort to make JavaScript standardization even more complete and effortless, have gone one step further. Why not use your server-side programming language to generate the JavaScript for you? That is, if you are using Ruby on Rails, perhaps you could write commands in Ruby and have them translated into JavaScript. Doing so would allow you to use roughly the same code in all of your templates, without having to switch syntax in different parts of the template.

This might sound like a strange idea, but the more I think about it, the more I like it. RJS (short for Ruby JavaScript) templates are one incarnation of this. If you prefer to create your JavaScript in Java, you might want to look at the Google Web Toolkit, which is now available under an open-source license and has gained many fans in the Java world.

This month, we look at RJS and how it makes life much easier for Web developers. Although I don't think that JavaScript will ever disappear, or that Web developers will be able to ignore it completely, technologies such as RJS mean that it might become something like machine code today—available and sitting at the bottom of the pyramid, but generally ignored by high-level programmers.

Listing 1. index.rhtml

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
    <head>
        <title id="title">Sample HTML page</title>
        <%= javascript_include_tag 'prototype' %>

        <script type="text/javascript">
        function updateHeadline()
        {
                var headline = $('headline');
                var new_headline_text = $F('future-headline');
                Element.update(headline, new_headline_text) ;
        }
        </script>

    </head>

    <body>
        <h1 id="headline">Headline</h1>
        <p><input type="text" id="future-headline" /></p>
        <p><input type="button" onclick="updateHeadline();"
                    value="Update headline" /></p>
    </body>
</html>
```

Ajax and Rails

To create a new Rails project called ajaxdemo, type:

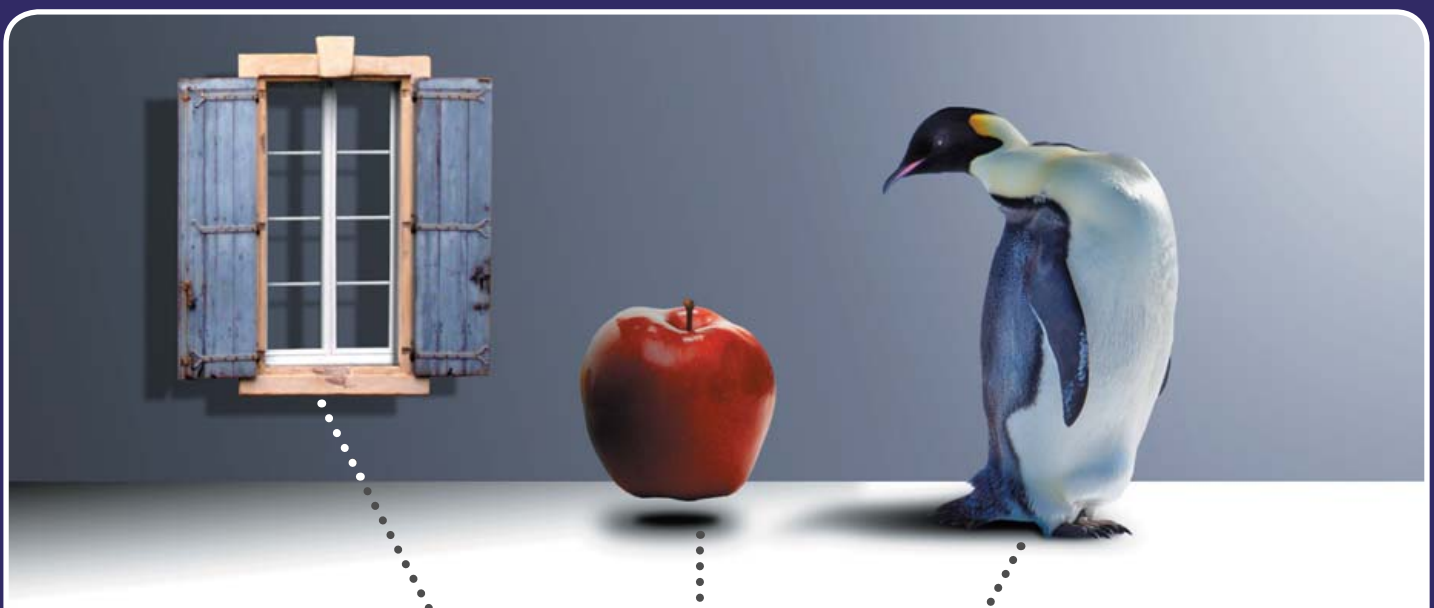
```
rails ajaxdemo
```

Now, let's create a simple controller called showme:

```
script/generate controller showme
```

We're not going to have any model in this system, but you still might have to define one or more lines in config/database.yml. Instead, let's create a new view within our showme controller, stored as app/views/showme/index.rhtml, shown in Listing 1.

As you can see, the index.rhtml page is a relatively standard page of HTML, with some JavaScript code that uses the Prototype library. The page consists of a headline, a text field and a button. Pressing the button invokes the



Your integrated neighborhood
just got friendlier

NAS 3004i
4TB Raw SATA Storage

NAS 3008i
8TB Raw SATA Storage

NAS 3012i
12TB Raw SATA Storage

>>>>> *From $7899* <<<<<

With Pogo's Network Attached Storage appliance, maximum interoperability with the most popular operating systems is easier. Take control of your ever-expanding storage needs with an easy to use graphical management interface, multiple snapshots, built-in back up, IP failover, and enhanced security. Powered by Intel Dual-Core Xeon processors and a fast 64-bit OS. Call today for more information.

www.pogolinux.com

pogo [linux]™

Experience, Imagination, and Support.
Pogo Linux is Hardware Solutions Built for Linux, Built for You.

To get started, contact us at 888.828.POGO or inquiries07@pogolinux.com
Pogo Linux, Inc. 701 Fifth Ave. Suite 6850, Seattle, WA 98104

intel Xeon® inside™
Quad-core.
Unmatched.

Even better, because we're returning a JavaScript program, rather than the contents of an individual HTML element, we can modify multiple parts of the page and even throw in some Scriptaculous effects for good measure.

function updateHeadline. This function takes the current value of the future-headline text field and changes the headline to reflect its contents.

Using a Remote Call

So far, we haven't done anything special. Now, however, we're going to do something a bit more sophisticated: send the contents of our text field to the server in an Ajax call. The server's response will be our headline, translated into Pig Latin.

Making this change requires doing two things. First, we need to write a method in our application controller that takes the headline, turns it into Pig Latin, and then returns that text. Second, we need to modify our template so that it gets the updated text from the server, rather than from a local JavaScript function.

Our updated template is shown in Listing 2. We have made a number of changes, starting with the fact that our form now has an id attribute associated with it, named theForm. The form contains a single element, a text field whose name is future_headline. Note that we need to use the name attribute instead of the id attribute, so that the form element will be submitted with our Ajax call. Also notice that we have changed the name to a Ruby-friendly future_headline (with an underscore), rather than the CSS-friendly future-headline (with a hyphen).

Listing 2. index.rhtml (Ajax Version)

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
    <head>
        <title id="title">Sample HTML page</title>
        <%= javascript_include_tag 'prototype' %>
    </head>

    <body>
        <h1 id="headline">Headline</h1>

        <form id="theForm">
        <p><input type="text" name="future_headline" /></p>
        </form>

    <p><%= submit_to_remote "submit-button",
                    "Pig Latin it!",
            :url => { :action => "piglatin_sentence" },
            :submit => "fakeForm",
            :update => "headline" %></p>
    </body>
</html>
```

We also have replaced our button with a call to the submit_to_remote helper:

```
<p><%= submit_to_remote "submit-button",
                "Pig Latin it!",
        :url => { :action => "piglatin_sentence" },
        :submit => "fakeForm",
        :update => "headline" %></p>
```

The above code does quite a few things:

- It creates a button, whose DOM ID is submit-button.
- The button has a label of "Pig Latin it!".
- When the button is clicked, it uses Ajax to invoke the piglatin_sentence action on the server, within the current controller.

Listing 3. showme_controller.rb

```
class ShowmeController < ApplicationController

  def piglatin_sentence
    # Get the headline
    sentence = params[:future_headline]
    words = sentence.split

    sentence = ""

    # Go through each word, applying the secret
    # Pig Latin algorithm
    words.each do |word|
      if word =~ /^[aeiou]/i
        word << "way"
      else
        first_letter = word.slice(0,1)
        rest = word.slice(1..-1)

        word = "#{rest}#{first_letter}ay"
      end

      sentence << word
      sentence << " "
    end

    render :text => sentence
  end

end
```

Are you shocked by the high cost of iSCSI & Fibre Channel storage?

AoE is your answer!

ATA-over-Ethernet = simple, low cost, expandable storage.

www.coraid.com

EtherDrive® SR1520

- RAID enabled 3U appliance with 15 slots for hot swap SATA disks
- Check out our other Storage Appliances and NAS Gateway

1. Ethernet Storage – without the TCP/IP overhead!
2. Unlimited expandability, at the lowest possible price point!!
3. You want more storage…you just buy more disks – it's that simple!!!

Visit us at www.coraid.com for more information.

CORAID

1.706.548.7200 The Linux Storage People **www.coraid.com**

Listing 4. index.rhtml (Updated for JavaScript in the HTTP Response)

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
    <head>
        <title id="title">Sample HTML page</title>
        <%= javascript_include_tag 'prototype' %>
    </head>

    <body>
        <form id="fakeForm">
            <h1 id="headline">Headline</h1>
        <p><input type="text" name="future_headline" /></p>
        </form>

    <p><%= submit_to_remote "submit-button",
                    "Pig Latin it!",
            :url => { :action => "piglatin_sentence" },
            :submit => "fakeForm" %></p>
    </body>
</html>
```

Listing 5. showme_controller.rb (Updated to Return JavaScript)

```
class ShowmeController < ApplicationController

  def piglatin_sentence
    # Get the headline
    sentence = params[:future_headline]
    words = sentence.split

    sentence = ""

    # Go through each word, applying the
    # secret Pig Latin algorithm
    words.each do |word|
      if word =~ /^[aeiou]/i
        word << "way"
      else
        first_letter = word.slice(0,1)
        rest = word.slice(1..-1)

        word = "#{rest}#{first_letter}ay"
      end

      sentence << word
      sentence << " "
    end

    output = "Element.update($('headline'), '#{sentence}');"
    render :text => output, :content_type => "text/javascript"
  end

end
```

- The contents of the form with the ID fakeForm are submitted.
- The value returned from the Ajax invocation is used to update the contents of the HTML element with the ID headline.

All that's left for us to examine is our controller, shown in Listing 3. The controller doesn't necessarily know that it is being invoked by a background Ajax process or that its contents will be used to update the headline element. Rather, it simply is invoked like any method, turning the words into Pig Latin. The translated sentence is returned to the user's browser as a plain-text file.

Returning JavaScript

Now, the real magic begins. As we just saw, our controller (piglatin_sentence) returns a plain-text document to its caller. Of course, we're free to return data in whatever format we please. One possible format might be XML. Indeed, the term Ajax is supposed to stand for Asynchronous JavaScript and XML, so it should come as no surprise that XML is a common format for return values. Another format that is growing in popularity is JSON (JavaScript Object Notation), a textual version of JavaScript objects that makes it fast and easy to exchange data.

But, in the world of Ruby on Rails, there is another type of data that the controller might return to the user's browser, namely JavaScript. This might not sound all that clever, but consider what Prototype does with it. If a controller is invoked with link_to_remote or submit_to_remote, and if the HTTP response has a content-type of xml+javascript, the JavaScript is evaluated.

This could potentially save time—instead of returning the text that should be used for the headline and then using JavaScript to insert it. (True, we were able to say this tersely by using an :update parameter to the call to submit_to_remote. But the code still exists.) Rather, we simply could return JavaScript code that uses the DOM to modify the document. This would simplify our code quite a bit.

To see this in action, look at Listings 4 and 5. Listing 4 is an updated version of our template, index.rhtml, and it is basically unchanged from the previous version, except that we now are able to remove the :update parameter from the call to submit_to_remote:

```
<p><%= submit_to_remote "submit-button",
                "Pig Latin it!",
        :url => { :action => "piglatin_sentence" },
        :submit => "fakeForm" %></p>
```

Instead of indicating what should be changed on the client side, we instead do that on the server side:

```
output = "Element.update($('headline'), '#{sentence}');"
render :text => output, :content_type => "text/javascript"
```

In other words, we tell our controller to produce a

response of type text/javascript, knowing that whatever we send will be evaluated by the user's browser. We then send a response that uses Element.update to change the headline to our translated sentence. Sure enough, as soon as we install this new version of our software, the headline continues to be changed.

The power that's associated with this is tremendous. For example, we could update the headline conditionally, checking it against a dictionary of forbidden words in our server's database. We could keep track of what words are most commonly used. We could restrict users to a certain number of headline updates per day.

Even better, because we're returning a JavaScript program, rather than the contents of an individual HTML element, we can modify multiple parts of the page and even throw in some Scriptaculous effects for good measure. Returning JavaScript is a seemingly simple feature that Prototype provides, but it is one that opens the door to tremendous possibilities.

RJS

And now, the moment you've been waiting for—you might be thinking that although the notion of evaluated JavaScript responses is powerful, it's annoying to have to create and maintain JavaScript code in Ruby controllers. It's bad enough that we need to have SQL in there; three languages in a single file seems like overkill.

And, that's where RJS templates come in. The basic idea is that we assume a response will be in the form of JavaScript, and that it will modify one or more elements on the current page. RJS provides us with a compact syntax for making those changes, so we can create very small files that do a great deal.

The changes we need to make are minor. First, we modify our piglatin_sentence method such that it modifies not sentence (a local variable) but @sentence (an instance variable). We also remove the call to render, because we won't be rendering anything directly.

We then create a file called piglatin_sentence.rjs. This is a view, just like an .rhtml file, and thus goes alongside it in the views directory. But, it consists of a single line:

```
page[:headline].replace_html @sentence
```

In other words, we should take the current page, find the element with an ID of headline and replace its HTML content with the value of @sentence, which we got from the method.

Sure enough, this works well. With a tiny bit of code, we've managed to do quite a bit. And, as before, we can add Scriptaculous calls, update multiple elements on the page, show and hide HTML elements—basically, anything we might want to do.

Listing 6. showme_controller.rb (Updated to Use RJS)

```
class ShowmeController < ApplicationController

  def piglatin_sentence
    # Get the headline
    sentence = params[:future_headline]
    words = sentence.split

    @sentence = ""

    # Go through each word, applying the
    # secret Pig Latin algorithm
    words.each do |word|
      if word =~ /^[aeiou]/i
        word << "way"
      else
        first_letter = word.slice(0,1)
        rest = word.slice(1..-1)

        word = "#{rest}#{first_letter}ay"
      end

      @sentence << word
      @sentence << " "
    end
  end

end
```

Conclusion

When I first read about RJS templates, I thought it was one of the weirdest ideas I had heard of. That's because the notion of writing JavaScript in Ruby seemed both strange and unnecessary. I now understand the power and cleverness of this type of template and look forward to using it on many of my Ajax-powered sites. With a bit of study and some changes in how you think about Web development, you're likely to make the same discovery.■

Reuven M. Lerner, a longtime Web/database consultant, is a PhD candidate in Learning Sciences at Northwestern University in Evanston, Illinois. He currently lives with his wife and three children in Skokie, Illinois. You can read his Weblog at altneuland.lerner.co.il.

Resources

There are many good printed and on-line resources for learning about Ajax. One of my favorite sites is **ajaxian.com**, in which the authors discuss and review Ajax-related tools.

Two good books might come in handy. The Pragmatic Programmers have published a second edition of the award-winning *Agile Development in Rails* by Dave Thomas and David Heinemeier-Hansson, and O'Reilly has released *Ajax on Rails* by Scott Raymond. Between these two books, you should expect to get a full understanding of not only how Ruby on Rails works, but also how to use JavaScript and RJS to create interesting and dynamic Web applications.

Languages Build Character, or Vice Versa

MARCEL GAGNÉ

Just what do you have to do to type those special characters anyhow?

If, like me, you spend a lot of time working with OpenOffice.org's word processor, Writer, the solution is fairly simple.

François, our guests will be here soon. What are you doing? *Quoi?* You are learning a language for this issue's theme? You might have picked a better time to study, *mon ami*, but I applaud you nonetheless. What language did you choose to learn? PHP? C++? Python? You chose Spanish? I am sorry, François, I am not laughing at you, I am just laughing in general. This issue isn't supposed to be about human languages; it's about computer languages. Now, don't fret. Let's hear a line of Spanish. You speak Spanish very good. Did you learn it from a book? Now, now, François, I promise. No more *Fawlty Towers* jokes. Besides, our guests will be here any moment. Head down to the cellar and bring back the 2003 Errazuriz Don Maximiano Cabernet from Chile. I'll finish getting the tables ready. *Vite!*

Ah, welcome everyone, to *Chez Marcel*, home of exquisite wines and super open-source software. Please, sit and make yourselves comfortable. François has gone to fetch the wine and will be back shortly. I have decided to make some last-minute changes to the menu in honor of my faithful waiter's well-meaning intentions. Tonight's menu will feature languages, or at least, dealing with the special characters that make up many languages.

Even if you don't speak any language other than English, there will be times when you find yourself needing to enter a special accented letter or character into your writing. This is particularly true if your name contains a letter with an accent at the end of it, as my own last name does. Sure, I could keep a document with these letters already written, handy and open on my desktop, select the letters, copy them and finally, paste that é into my text, but doing so can become amazingly time consuming.

If, like me, you spend a lot of time working with OpenOffice.org's word processor, Writer, the solution is fairly simple. Click Insert on the menu bar and select Special Characters from the drop-down menu. A window labeled Special Characters appears (Figure 1).

From the dialog, you can select your font and international character set—North and South Americans, Australians and most Europeans will work with the Latin subset. All in all, this is great solution, except that it works only with OpenOffice.org running and requires you to open the dialog every time you want to enter a special character. Furthermore, not all applications have a handy list of characters from which to choose. There are several ways to get around this problem, and I show you a few of them now.

Let's start with some KDE solutions. Fire up kcontrol, the KDE Control Center. Under Regional & Accessibility, select the submenu for Keyboard Layout. On the right-hand pane, you'll see a number of flags from different countries. Click Enable keyboard layouts to un-gray these choices. If you installed your system using US English as your language of choice, the default is to provide you with US English. Because my background is French Canadian rather than French from France, I am most familiar with a North American keyboard layout, including the French Canadian keyboard (I honestly have no idea how a French keyboard in France is laid out). Consequently, I always add Canada to that list (Figure 2).

When you click OK or Apply, a little icon that looks like a tiny flag appears in your icon tray. Clicking that icon switches between the various layouts (of which there are

Figure 1. OpenOffice.org provides a dialog for entering special characters into your documents.

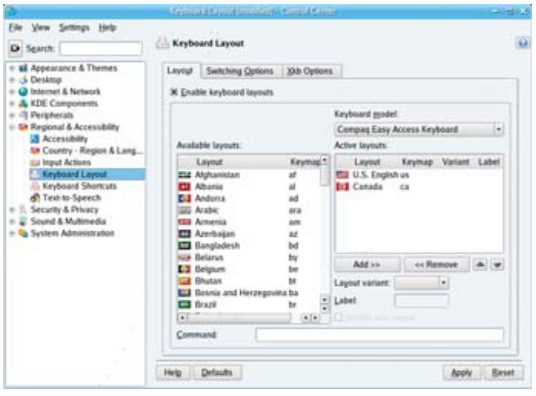


Figure 2. From the KDE Control Center, you can assign additional keyboard layouts.

two at the moment). When you are on the alternate layout (for example, Canada), pressing different keys causes whatever character from the keyboard layout that the map presents to appear. This may require some experimentation on your part so you can discover what each key does—unless, of course, you happen to be familiar with your chosen layout.

In order to continue working normally, click the tray icon another time to switch back to the US English keyboard layout. This switching back and forth, however, isn't for everyone, and taking the time to learn an alternate keyboard layout when you need only the occasional character may not be the best approach. Another way around this problem is to use a tool called KCharSelect. You generally can find it in your KDE Utilities menu, but you also can run it with the command `kcharselect`. When the program starts, a table of all 256 available characters in your current locale appear (Figure 3).

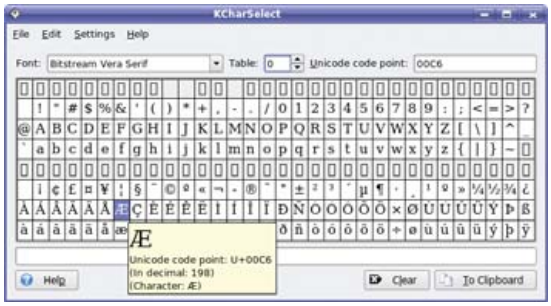


Figure 3. KCharSelect is like a buffet table for special characters.

Select a font from the drop-down list at top. Then, if necessary, select an alternate character table (the Latin set is at 0 on my system). If you hover your mouse pointer over a character, you'll see its Unicode and ASCII value. To use a character in an application (such as e-mail), click on that character, and it will appear in the text field at the bottom of the dialog. You even can enter multiple characters if you want. To use what you have selected, click the To Clipboard button, and then paste the result into your application of choice.

A similar application called Gucharmap exists for users of the GNOME desktop. You likely can find it under the Accessories submenu by clicking the Applications button on the top GNOME panel (as it is under my Ubuntu test system). The program name is `gucharmap` if you want to run it directly. When the window appears (Figure 4), you'll see that the concept is similar to the KDE KCharSelect tool, but there are some interesting differences. From the left-hand sidebar, select your character set or script (Latin, for most of us here), choose a font, then double-click on the character you want from the main display pane on the right. When you do so, that character appears in the Text to copy field at the bottom of the dialog. Click the Copy button, then paste the text into whatever application you are currently running.

Tools like KCharSelect and Gucharmap are wonderful

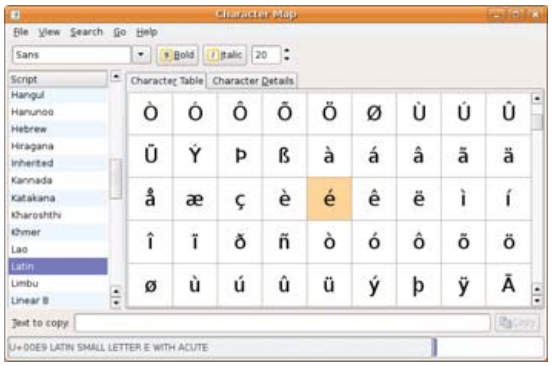


Figure 4. Gucharmap is a great GNOME way to select special characters.

to have at your disposal, but for some people, there is only an occasional need for entering special characters. As such, these programs seem to live a little large on the desktop, *non?* A tiny application, an applet that lives in your panel, might be more to taste for those of us who need to enter only a small number of characters. In the KDE environment, there are many applets, and one of these, the Character Selector, may be just what you need.

To add the applet, right-click on a blank section of your Kicker panel, and select Add Applet to Panel. The Add Applet dialog appears with a list of programs you can use to populate your Kicker panel (Figure 5). Each program (or applet) is listed alphabetically with a short description. Look for the Character Selector. To add the applet, simply click the Add to Panel button.

Figure 5. Adding applets to the KDE Kicker panel.

Your Kicker panel now should have a box containing a handful of special characters—12 by default. To use one of these characters in your text, simply single-click the character of your choice, and paste it into your application. It's that easy. That said, the set of characters provided by the application may not be quite what *you* want. To change that, right-click on the applet's handle (that little bar with the arrow to the left of the applet), and select Configure Character Selector from the pop-up menu. A configuration window appears above the applet (Figure 6).

Tools like KCharSelect and Gucharmap are wonderful to have at your disposal, but for some people, there is only an occasional need for entering special characters.

Figure 6. By adding the Character Selector to your Kicker panel, special characters are always handy.

Modify the character list to suit your taste or need, then click OK. You may want to use KCharSelect, just this once, to paste in your list of characters.

Those of you who are running GNOME as your default desktop environment have a similar tool at your disposal, and it also is an applet. Right-click on the bottom (or top) panel and select Add to Panel. A window of the same name appears with a list of available applets (Figure 7). Scroll down until you see the Utilities section. There you will find an applet named Character Palette. Select it, then click Add or simply drag the applet to the panel in the location of your choice.

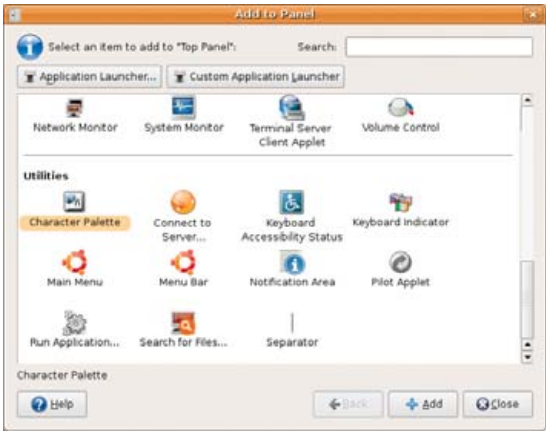


Figure 7. Add the GNOME Character Palette panel applet.

You will see a row of characters appear on your panel with a drop-down arrow to the left of those characters. To use any of these characters in a document (or e-mail message or chat session or other application) click the character of your choice, then paste it into your application. If you don't see the character you need, there's a good chance it already has been defined for you. All you need to do is switch palettes. Click the down arrow, and a selection of more than 20 predefined palettes appear; simply click to switch. As rich as this selection is, it still is possible that the characters you need on a day-to-day basis are not there. To edit the palette or add to it, right-click on the down arrow and select Preferences from the submenu. The Character Palette Preferences dialog appears (Figure 8).

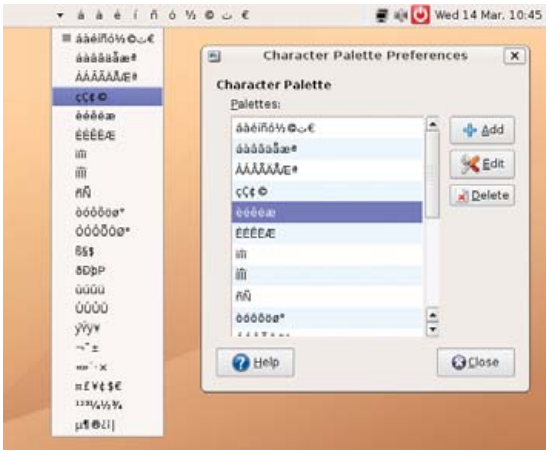


Figure 8. So many special characters, so little time.

From this dialog, you can choose to edit a current palette, delete a palette or create something entirely new. To create a new palette, click the Add button and enter the characters into the dialog window that appears—of course, you may need to paste them from another character application, such as Gucharmap or OpenOffice.org.

Now, when François becomes a master of Spanish, he will be able to enter all the Spanish characters he wants. In the meantime, *mes amis*, it appears that it is nearly time to bid you all *Adieu*. Nearly, being the key word, or as François might now say it, *casi*. *Quoi?* You plan on learning Japanese next week? Why not. Until next time, please raise your glasses, *mes amis*, and let us all drink to one another's health. *A votre santé! Bon appétit!*■

Marcel Gagné is an award-winning writer living in Waterloo, Ontario. He is the author of the all-new *Moving to Free Software*, his sixth book from Addison-Wesley. He also makes regular television appearances as Call for Help's Linux guy. Marcel is also a pilot, a past Top-40 disc jockey, writes science fiction and fantasy, and folds a mean Origami T-Rex. He can be reached via e-mail at mggagne@salmar.com. You can discover lots of other things (including great Wine links) from his Web site at www.marcelgagne.com.

Resources

GNOME Web Site: **www.gnome.org**

Gucharmap: **live.gnome.org/Gucharmap**

KDE Web Site: **www.kde.org**

OpenOffice.org: **www.openoffice.org**

Marcel's Web Site: **www.marcelgagne.com**

The WFTL-LUG, Marcel's On-line Linux User Group: **www.marcelgagne.com/wftllugform.html**

SUPERMICRO®

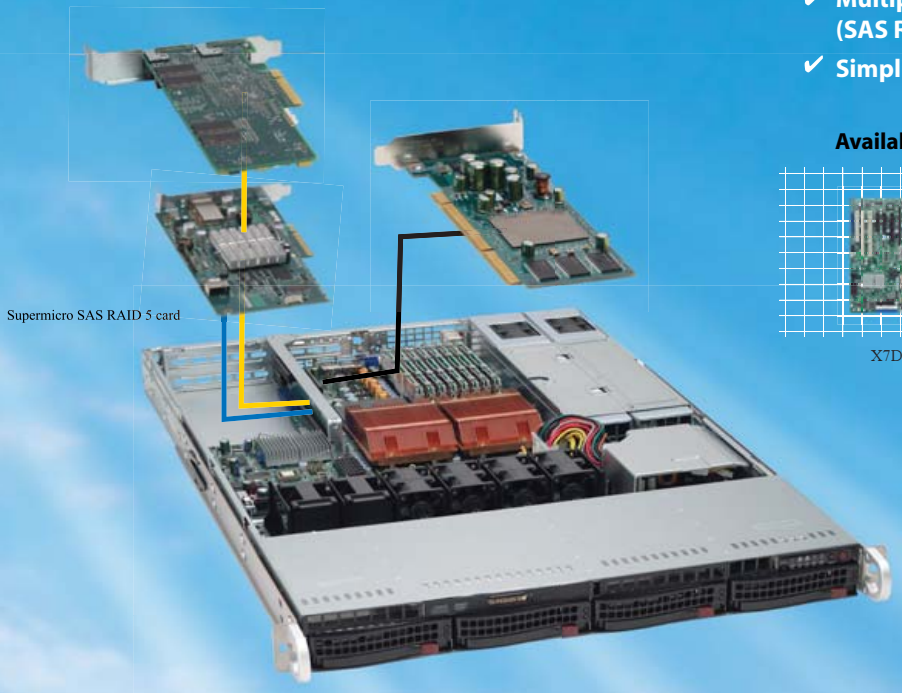
UIO

Universal I/O

Maximized I/O Expandability & Flexibility

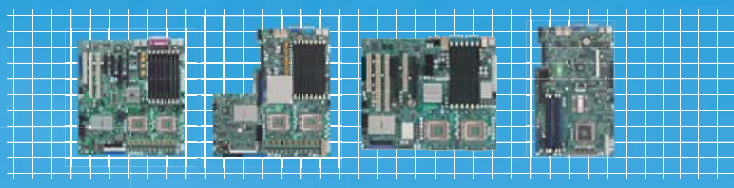
The Freedom to Build Exactly What You Need.

- ✔ 3/7 Add-on Cards for 1U/2U
- ✔ Highly Upgradeable
- ✔ High Efficiency Power (up to 90%+)
- ✔ Multiple Expansion Card Options (SAS RAID 5, 10-G, IB...up to 20 choices)
- ✔ Simplify Your Inventory Management



Supermicro SAS RAID 5 card

Available Motherboard Options



X7DBN | X7DBU | X7DAL-E+/X7DAL-E | PDSMU

Optimized Chassis Solutions

SC815TQ-R650U | SC825TQ-R700U

Supermicro UIO servers allow users to select from a wide range of I/O options to provide the ultimate in storage and networking flexibility. The UIO card becomes a part of the serverboard, allowing the system to retain all of it's PCI Express and PCI X slots for expansion cards. As a result, future upgrades can be acheived by replacing the UIO card and/or expansion cards instead of replacing the entire system. This versatility helps to minimize the amount of different servers that customers need to operate their business.

intel® Xeon® inside™ TM

Quad-core. Unmatched.

For more information visit us at www.supermicro.com

* Our new generation power supply efficiency measures 90%+ or more under a typical loading operation.

AMAX	Arrow Electronics	ASI	Bell Micro	Ingram Micro	MA LABS	Synnex	Tech Data
1-800-800-6328	1-888-427-2250	1-800-2000-ASI	1-800-232-9920	1-800-456-8000	1-408-941-0808	1-800-756-5974	1-800-237-8931
www.amax.com	www.arrownacp.com	www.asipartner.com	www.bellmicro.com	www.ingrammicro.com	www.malabs.com	www.synnex.com	www.techdata.com

DAVE TAYLOR

Displaying Image Directories in Apache, Part III

The incredible shrinking script knows a better way to resize your thumbnails.

In last month's column, we built our directory display script to the point where you could get a smart listing that showed your image files (offering links to any other file type), and we allowed thumbnails to be displayed too.

The latter trick is done by letting the Web browser do the work. If you specify a height or width that's different from the actual image size, Web browsers automatically scale the image to fit the specified dimensions. Even better, if you specify only one dimension, it scales proportionally to fit.

Let me explain that just a wee bit more, because it's critical to this particular scripting project. If you have an image that's 250x250 pixels and you'd like to display a 75x75 thumbnail, the best practice is to specify both height="75" and width="75", of course. The problem is, what if the image is actually 250x317 and you want to reduce it to exactly 75 pixels wide. How tall should it be?

You could do the math, of course, but it's much nicer to let the browser do the work for you automatically, which happens if you specify only width="75" or use a full HTML statement:

```
<img src="my250x317.png" width="75" />
```

Doing that scales it, and you end up with an image that's exactly 75x95 pixels in size. However, if you always constrain one dimension, things can break. What if the image is actually 250x1100, because it's a very tall graphic? Now the thumbnail is going to break the entire layout, because the scaled version of it is 330 pixels wide, quite a bit more than the 75x75 target box for the image!

That's why an ideal script would figure out which of the dimensions is larger, and then constrain that one to the size of the box we seek, letting the other scale proportionally automatically, thanks to the Web browser. And, that's exactly what we'll do!

Big Important Caveat: I realize there's a significant performance penalty for letting the browser scale images—the entire full-size image has to be downloaded, even though you're seeking a smaller version. If it was a problem, you could use a tool such as ImageMagick to scale the images and create thumbnail graphics that were displayed instead, probably dropping them into a cache and creating new ones on the fly as needed. But honestly, don't you have a high-bandwidth Internet connection, and does an additional second or two of load time really matter?

On to the Script!

Last month, we created the darn useful script function figuresize, which, when given a graphic image, returned height and width parameters when those could be calculated. The resultant main loop in the script ended up looking like this:

```
for name in *
do
  if [ ! -z "$(file -b $name|grep 'image data')" ]
  then
    figuresize $name
    if [ ! -z "$height" ] ; then
      echo "<img src=$name alt=$name height=50 />"
      echo "<br />$name ($height x $width)<br />"
    else
      echo "<img src=$name alt=$name height=50 />"
      echo "<br />$name<br />"
    fi
  else
    echo "<a href=$name>$name</a><br /><br />"
  fi
done
```

If you read the code closely, it's really not doing anything smart with the height and width parameters, just displaying them in the output. Instead, let's turn that into a test to figure out which is larger. Before I do that though, we need to make some rudimentary improvements to the loop so the output is more attractive:

```
for name in *
do
```

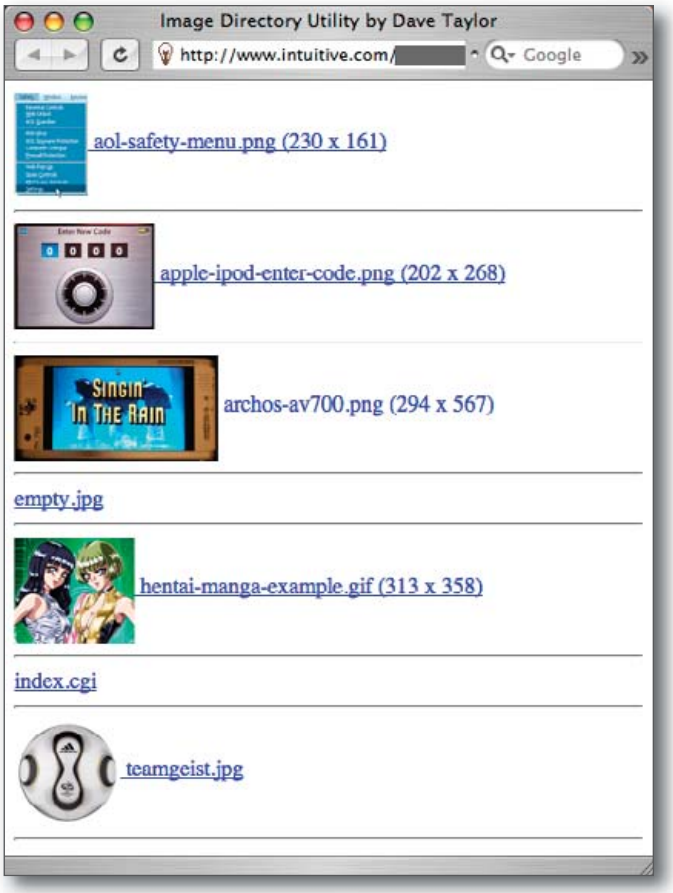


Figure 1. Result of Running the Improved Script

```
  if [ ! -z "$(file -b $name|grep 'image data')" ]
  then
    figuresize $name
    if [ ! -z "$height" ] ; then
      echo "<a href=$name><img src=$name border=0"
      echo "alt=$name height=$size "
      echo "align="absmiddle" />"
      echo "$name ($height x $width)</a>"
    else
      echo "<a href=$name><img src=$name border=0"
      echo "alt=$name height=$size"
      echo "align="absmiddle" />"
      echo "$name</a>"
    fi
  else
    echo "<a href=$name>$name</a><br />"
  fi
  echo "<hr />"
done
```

The result of running this improved script (where images are clickable, there's a horizontal rule between entries and so forth) is shown in Figure 1.

Now, let's look at how to make the script even smarter:

```
if [ ! -z "$height" ] ; then
  if [ $height -gt $width ] ; then
     dimensionlabel="height"
  else
     dimensionlabel="width"
  fi
```

Can you see what I've done here? This lets us figure out which of the two dimensions of the graphic is larger and then set the dimensionlabel to that particular dimension. Here's the result:

```
echo "<img src=$name $dimensionlabel=$size />"
```

where I'll set size to the desired thumbnail size—75 in our example script.

I'm also going to add a few counters so we can summarize images displayed versus total files displayed at the end. Just because it's, uh, interesting, right?

Here's the latest version of the loop, and as you might expect, it's getting more complicated as it becomes more sophisticated:

```
for name in *
do
  if [ ! -z "$(file -b $name|grep 'image data')" ]
  then
    imgcount=$(( $imgcount + 1 ))
    figuresize $name
    if [ ! -z "$height" ] ; then
      if [ $height -gt $width ] ; then
        dimensionlabel="height"
      else
        dimensionlabel="width"
      fi
      echo "<a href=$name><img src=$name border=0"
      echo "alt=$name $dimensionlabel=$size"
      echo "align="absmiddle" />"
      echo "$name ($height x $width)</a>"
    else
      echo "<a href=$name><img src=$name border=0"
      echo "alt=$name height=$size"
      echo "align="absmiddle" />"
      echo "$name</a>"
    fi
  else
    echo "<a href=$name>$name</a><br />"
  fi
  echo "<hr />"
  totcount=$(( $totcount + 1 ))
done
```

Figure 2. More Attractive Output

```
echo "<i>Displayed $imgcount images out of
$totcount entries total.</i>"
```

The resultant output, which is hopefully more attractive, is shown in Figure 2.

Now that we can normalize these thumbnails in the script (at least for non-JPEG images, due to a limitation in the file command), the next thing to examine is how to display the results with multiple images across, in a grid or table, rather than one per line as we see now. That's a bit more complicated, because it involves yet another counter, but while you're waiting for your next issue of *Linux Journal*, you might bone up on the basic HTML table tags, because that's what we'll be using. Then, finally, we'll switch to ImageMagick from file, so we can get the dimensions of all image files, not only GIF and PNG files.■

Dave Taylor is a 26-year veteran of UNIX, creator of The Elm Mail System, and most recently author of both the best-selling *Wicked Cool Shell Scripts* and *Teach Yourself Unix in 24 Hours*, among his 16 technical books. His main Web site is at www.intuitive.com, and he also offers up tech support at AskDaveTaylor.com.

The Straight Talk People℠
SINCE 1991

ABERDEEN

STORAGE

ABERDEEN STONEHAVEN X318

3U 12TB Dual-Core Ready Storage Server

- Up to two Dual-Core AMD Opteron™ 2000 Series processors
- Up to 16 x 750GB (12TB) Hot-Swap SATA Hard Drives
- Internal SATA 2.5" Hard Drive Bay for OS Drive
- 800+ MB/sec sustained data throughput RAID Controller
- Up to 32GB 667/533MHz ECC Registered DDR2 SDRAM
- nVIDIA nForce Pro Chipset with 64-Bit Support
- 650W Redundant Hot-Swap Power Supply
- **5-Year Warranty**

Starting at **$4,259**

ABERDEEN STONEHAVEN X526

5U 18TB Dual-Core Ready Storage Server

- Up to two Dual-Core AMD Opteron™ 2000 Series processors
- Up to 24 x 750GB (18TB) Hot-Swap SATA Hard Drives
- Two Internal SATA Hard Drive Bays for Mirrored OS Drives
- 800+ MB/sec Sustained Data Throughput RAID Controller
- Up to 32GB 667/533MHz ECC Registered DDR2 SDRAM
- nVIDIA nForce Pro Chipset with 64-Bit Support
- 950W Triple Redundant Hot-Swap Power Supply
- **5-Year Warranty**

Starting at **$5,679**

ABERDEEN STONEHAVEN X633

6U 24TB Dual-Core Ready Storage Server

- Up to two Dual-Core AMD Opteron™ 2000 Series processors
- Up to 32 x 750GB (24TB) Hot-Swap SATA Hard Drives
- Two Rear Hot-Swap SATA Hard Drive Bays for Mirrored OS Drives
- 800+ MB/sec sustained data throughput RAID Controller
- Up to 32GB 667/533MHz ECC Registered DDR2 SDRAM
- nVIDIA nForce Pro Chipset with 64-Bit Support
- 1350W Redundant Hot-Swap Power Supply
- **5-Year Warranty**

Starting at **$7,139**

"The Ultimate Linux Server... too fast for our benchmarks... we recommend the Aberdeen line of servers without reservation."
Linux Journal—Aberdeen Stonehaven A261T

"terrific for video serving or other storage intensive tasks"
PC Magazine —Aberdeen XDAS

"Aberdeen surpasses HP ... markedly higher scores ... AberNAS 128 boasts outstanding features"
Network Computing—Aberdeen AberNAS 128

"powerhouse performance ... staggering ... eye-opening ... the highest WebBench numbers to date"
PC Magazine—Aberdeen Stonehaven A261S

AMD 64 Opteron™

888-297-7409
www.aberdeeninc.com/linux

JON "MADDOG" HALL

Languages—Some Dead and Some Still Kicking

There's more to programming than Java.

"maddog", rang out the voice that I knew very well. It belonged to one of my young friends, Eduardo. "Why did you name your boat *Agape*? Is it because the boat is wide open?"

I stopped sanding my little sloop and faced my young friend. "No, the word is not English, but Latin. It is not meant to rhyme with a small fruit that is used in wine, but to speak of the highest form of love."

Eduardo looked at me a moment and said, "I did not know that you spoke any languages other than English."

"For the most part, I do not", I answered. "I studied Latin and French in grade school many decades ago, but by the time I understood how valuable it was to know multiple languages, most of that training had slipped away. Yet, having studied Latin and French does come in handy from time to time, as I sometimes use that forgotten training to understand new words both in foreign languages and my native English. Now, the languages I study are mostly computer languages."

"That is another thing", said Eduardo. "Why are there are so many languages when all you really need is Java?"

I put down my sanding block, leaned up against the *Agape* and thought carefully for a little while.

FORTRAN was the first computer language I ever learned. It was in 1969, and I learned FORTRAN by reading a book called *Programming the IBM 1130 in FORTRAN* and practicing on an IBM 1130 computer system that had 4,000 words of main memory and a card reader and punch. The system also had a chain printer (you probably do not want to know what that was) and a pen plotter. Notice that the name of the language was FORTRAN, in all capital letters. That is how we wrote it in that day, for it stood for FORmula TRANslator, just like COBOL stood for COmmon Business Oriented Language. FORTRAN was for engineers and scientists, and COBOL was for business people. Both languages did their jobs fairly well.

Because I was an engineering student and a co-op for the Western Electric Corporation (a member of the Bell System), I learned FORTRAN. Somewhere along the line, marketing people decided that people did not like all capital letters in names, so the language became called Fortran. Through the years, FORTRAN (which started in the early 1950s) became FORTRAN II, FORTRAN III, FORTRAN IV and then started to use names related to the years that it was updated. Today, work is being done for a definition for Fortran 2008. Fortran is a good example of a language (and not just a name) that changed to meet the needs of the time.

After returning to Drexel Institute of Technology (now Drexel University) from my co-op period, I started seeking out more of those devices known as computers. I found several Digital PDP-8 mini-computers in a computer lab of the Electrical Engineering Department. Although these systems had a small language that was FORTRAN-like, called Focal, the PDP-8 mini-computers at Drexel were mostly programmed in assembly and/or machine language, the ones and zeros that the machine used.

I was given some books on how to program the PDP-8 in machine language, and I taught myself how to program using the most fundamental language of the computer. Fortunately for me, the machine language of the PDP-8 was a very simple one, having only eight basic instructions and one main register that acted as an "accumulator". Each instruction was the same length and matched the word size of 12 bits, so the PDP-8 was simple, though tedious, to program. The PDP-8 could not subtract (much less multiply or divide), so you had to add the two's complement of the subtrahend in order to subtract.

The processors had switches and lights on the front panel, and by toggling these switches you could input the program directly into the memory of the system. More important, you could step through your program one machine language instruction at a time, seeing the results of each instruction on the accumulator of the machine and in the program counter (also designated by lights on the console).

Most people did not enter their entire program through the switches, of course; they used an ASR-33 Teletype to enter the source code of the assembly languages into an editor, punched out a source-code paper tape, input that tape to an assembler and (finally) got an object-level tape punched that would contain the 1s and 0s to be fed into the computer.

One turnaround of your program typically took 45 minutes at a minimum. It took five minutes to read in the editor (paper tape). Next, you typed in your program, did a few edits and punched out your program onto new paper tape. Then, you read in the three-pass assembler (15 minutes of paper-tape reading), read in your source-code program (assuming your paper tape did not rip) and punched out the binary (paper) tape that contained your program.

Finally, you read your binary paper tape into the memory of the computer, watched while the program probably over-wrote itself and everything else in real memory, and then you started the whole process over again

(after cursing loudly at the programming gods).

But, it was fun and a challenge. And, it was the "will of the machine" versus the pure logic of 1s and 0s. I was hooked.

During this same period, I switched from being an Electrical Engineer to a major that was both engineering and business, with a minor in what became Computer Science. I studied several different languages: Algol, Lisp, PL/I, SNOBOL and APL, each one with its own special niche in the computer field.

I remember SNOBOL as a language for string processing, and although I do not remember much of it, I do remember thinking that almost any string of characters input into a SNOBOL compiler would generate some type of syntactically correct program. I remember thinking that the only syntax error that might be generated was the lack of the END statement.

APL, on the other hand, was an array programming language that was very powerful. It even fostered its own special set of symbols that meant you had to paint them on the side of the keys of your regular keyboard, and (in the case of an IBM Selectric typewriter) use a print head specially made for APL.

In a class on comparative language design, our professor gave us a challenge of re-implementing a 40-line FORTRAN program in as few lines of APL as possible. Most of us reduced the FORTRAN program to three lines of APL, some to two lines of APL. And, one of the brightest students in the class (David Erb) had stayed up the entire night reducing the program to only one line of APL. It was a truly amazing line of code.

After inputting the data to the program and seeing the correct result, the professor (with a gleam in his eye) asked David to explain how the program worked. David, having finished the program only hours before, tried desperately to remember how it actually worked, but he could not explain how he had come to that particular single line, or even how it would produce the desired result.

As a class, we had experienced our first "write-only" language. It was a lesson remembered over the next 38 years—never write a program you could not easily change or re-use.

It was an interesting time in the days before Computer Science degrees. It was more like "Computer Black Magic", where the computers were owned by the Math Department, the Electrical Engineering Department, the Business school, the Physics Department and so forth. I even had one professor who told me I never would be able to make a living writing software. It has yet to be

seen if he was correct.

And, as time went on and computers became more and more important to the world, universities started granting Computer Science, Computer Engineering, Network Engineering and Information Science degrees. And, what we experienced as very painful steps toward understanding these procedures became formalized into what we now call (generally) Computer Science.

I eventually graduated from Drexel University, and in looking for my first job, I was determined that I would not program in a "higher-level language", such as FORTRAN or COBOL. I wanted to program in assembly language. I turned down many jobs, looking for that chance.

Eventually, that chance came to me: to program in the systems programming group of Aetna Life and Casualty in Basic Assembly Language (BAL) for the IBM 360 series computer—an assembly language and computer system I had never seen before. They asked me if I could program it in assembly language. I said, "Sure, just show me the book."

Fortunately there was a book, *Programming the IBM 360 in BAL*, and after reading it and practicing a few days at Aetna, I started a four-year career of system programming there—a time in my life where I learned many, many things.

The one "language", if I may call it that, which sustained me over the years, however, was the 1s and 0s of machine code.

It was machine code that allowed me to learn how compilers and interpreters can translate different languages to what the machine can follow. It was machine code that showed me how reentrant and recursive languages worked, and how minute changes in the code of a program could cut many minutes, if not hours or days, off a program's execution.

It was the study of machine code and machine architecture that allowed me to know how the operating system really worked, and allowed me to understand the protocols of networking. And, it was machine code and looking at the sources of it that let me understand the issues of big-endian versus little-endian and single-precision versus double-precision.

And, it was the knowledge of machine language that allowed me to find places where the compilers had made mistakes in doing the translation of source code to those 1s and 0s. People who knew only higher-level languages could have kept looking at the source code of those high-level languages forever and would not have found the problems.

Nevertheless, other than brief excursions into teaching machine and assembly language courses (the last time was in 1985 when I taught a course in PDP-11 assembly language, another assembly language that I taught myself), I have not actually written in assembly or machine language since I left Aetna in 1977.

The simple reason is that it takes too long to write in those low-level languages, and it is too error-prone. Compiler optimization techniques have gotten better, CPUs faster, memories cheaper and people's time more expensive. Don't get me wrong; there still are plenty of places where assembly language and machine code are used, particularly in places that need absolutely the best performance or the smallest size. But overall, the compilers are doing a pretty good job, and the average person's mind does not adapt well to the tedious task of writing in assembly.

Yet, when I do sit down and write a bit of code in some modern language, I do two things:

- I think about whether there is a language that can do the program better and in a clearer, more human-maintainable way.
- I think about what that language may be generating in machine language and whether changes to the source code or to the algorithm or to the way the data is positioned in memory might affect the speed at which the program runs.

A friend of mine, David Mossberger, once did a study of multiplying two very large arrays. One multiplication was done in the standard linear-algebra "textbook" fashion of analyzing rows and columns. The second multiplication took into account the cache of the CPU and memory, knowing how the 1s and 0s were arranged. The second multiplication took one-fortieth of the time on an Alpha processor (which had large cache memories), and one-tenth of the time on an Intel processor (with smaller cache memories).

During the years, I have been exposed to many new languages and many new technologies. I have been able to understand each and every one because I stop to think about what the 1s and 0s are doing. There have been few new computer technologies that have stumped me in the past 38 years.

This is why I object to colleges and universities who feel that high-level languages, such as Java (or Python, or PHP, or you name it) are the only languages worth teaching, and that machine and assembly languages are not worth teaching to students.

In the end, it all comes down to 1s and 0s, and if you do not know what they are doing, and how the machine actually works, you are at the mercy of others.

In a lot of ways, machine code and assembly language are like Latin—perhaps little used, but still useful to know for the serious language enthusiast and programmer.

Carpe diem from the *Agape*.■

Jon "maddog" Hall is the Executive Director of Linux International (www.li.org), a nonprofit association of end users who wish to support and promote the Linux operating system. During his career in commercial computing, which started in 1969, Mr Hall has been a programmer, systems designer, systems administrator, product manager, technical marketing manager and educator. He has worked for such companies as Western Electric Corporation, Aetna Life and Casualty, Bell Laboratories, Digital Equipment Corporation, VA Linux Systems and SGI. He is now an independent consultant in Free and Open Source Software (FOSS) Business and Technical issues.

Polywell's Ultimate Linux Systems

More Choices, Great Service, Best Value!

The newest & fastest technology available TODAY!!!

1U Value Servers Starts at $399, Up to 4GB RAM

(custom config. available)

$399 1U-485Ax Sempron™ 3000+, 512M DDR2, 80GB HD (For Volume Purchase Only)

$499 1U-485Ax Athlon™64 3500+, 1GB DDR2, 80GB HD (For Volume Purchase Only)

$599 1U-485Ax Athlon™64 X2 Dual-Core 3600+, 2GB DDR2, 2x80GB HD (Dual LAN +$45)

$799 1U-690GA Athlon™64 X2 Dual-Core 4200+, 4GB DDR2, 2x80GB HD (Dual GigaLAN +$45)

Linux Appliance Starts at $299

Poly 485Ax Sempron™ 3000+, 256M DDR2, ATI X1100 Graphics, 100Mbit LAN, DVD, 80GB HD $299 (For Volume Purchase Only)

Poly 690GA Athlon™64 3500+, 512M DDR2, ATI X1250 DVI+VGA, Gigabit LAN, DVD, 500GB HD $499 (For Volume Purchase Only)

(OEM /ODM Service Available)

1U Advanced Servers, Up to 64GB RAM, 4TB HD

$1,799 1U-690S4 Athlon™64 X2 Dual-Core 5200+, 4GB DDR2 2TB 4x500GB HD, Dual Gigabit LAN

$1,999 1U-1000SL Opteron™ 1210 Dual-Core, 8GB ECC, 2x80GB

$2,999 with 2TB 4x500GB HD, Opteron 1214

$4,599 1U-2500M 2 x Opteron™ 2212 Dual-Core, 16GB ECC DDR2, 2TB 4x500GB HD, Dual Gigabit LAN

$7,699 1U-2500M 2 x Opteron™ 2216 Dual-Core, 32GB ECC DDR2, 2TB 4x500GB HD, Dual Gigabit LAN (Option: 64GB+4TB HD)

Low-Cost NAS Storage 1.5TB Starts at $999

$999 Netdisk 4000A 1.5TB 3x500G

$1,199 Netdisk 4000B 2TB 4x500G

$1,799 Netdisk 6000A 3TB 6x500G (2U)

$4,599 Netdisk 12000B 6TB 12x500G (2U)

1U Twin Servers, 1U 8-Way Quad Opteron™

$6,499 1U-Twin 2 x 2 Dual-Core Processors, 2 x 8GB ECC DDR2, 2 x Dual 250GB HD, 2 x Dual Gigabit LAN

$7,299 1U-8415A 4 x Opteron™ 8212 Dual-Core, 16GB ECC DDR2, 1.5TB 3x500GB HD, Dual Gigabit LAN

$10,999 1U-8415S 4 x Opteron™ 8214 Dual-Core, 32GB ECC DDR2, 2x74GB 15K RPM SCSI HD, Dual Gigabit LAN

High-Density Multi-Processor Servers

2U 8-Way, 5U 16-Way Servers, Up to 128G RAM

$7,999 2U-8450A 4 x Opteron™ 8212 Dual-Core, 16GB ECC DDR2, 3TB 6x500GB HD, Dual Gigabit LAN

$11,500 2U-8450A 4 x Opteron™ 8214 Dual-Core, 32GB ECC DDR2, 3TB 6x500GB HD, Dual Gigabit LAN

$21,999 5U-8850T5U 8 x Opteron™ 8212 Dual-Core, 64GB RAM, 2TB 4x500GB HD, 3 x Gigabit LAN

$49,999 5U-8850T5U 8 x Opteron™ 8214 Dual-Core, 128GB RAM, 4TB 8x500GB HD, 3 x Gigabit LAN

9TB 2U Storage, 32GB RAM
2012SC 2500M-2220

9TB 12x750G, 32G, 2x2220 $13,999
6TB 12x500G, 4GB, 2x2210 $5,999

18TB 4U Storage, 64GB RAM
4024AIS 2500M-2220

18TB 24x750G, 64G, 2x2220 $36,999
12TB 24x500G, 4GB, 2x2210 $9,999

Blade Servers - 10 Dual or Quad Processors Blades

$17,999 8U 10 x 2055A Blades
10 x (Dual Opteron™ 2210 Dual-Core, 4GB RAM, 80G HD)

$39,999 8U 10 x 2500M Blades
10 x (Dual Opteron™ 2212 Dual-Core, 16GB RAM, 80G HD)

$64,999 8U 10 x 8450A Blades
10 x (Quad Opteron™ 8212 Dual-Core, 16GB RAM, 80G HD)

$99,999 8U 10 x 8450A Blades
10 x (Dual Opteron™ 2214 Dual-Core, 32GB RAM, 80G HD)

AMD Dual-Core technology enables one platform to meet the needs of multi-tasking and multi-threaded environments; provides platform longevity

- 20 Years of Customer Satisfaction
- Industry's Longest Warranty (5 years)
- Customizing available for all systems
- First Class Customer Service

Polywell OEM Services, Your Virtual Manufacturer
Prototype Development with Linux/FreeBSD Support
Small Scale to Mass Production Manufacturing
Fulfillment, Shipping and RMA Repairs

AMD 64 Opteron

AMD 64 Athlon X2

888.765.9686
www.Polywell.com/us/LJ

Polywell Computers, Inc 1461 San Mateo Ave. South San Francisco, CA 94080 650.583.7222 Fax: 650.583.1974

POLY POLYWELL

DOC SEARLS

Picking New Fights

Now that Linux has won, what's the next cause to take on?

That's why fighting for Linux today is like fighting for geology, botany or the periodic table.

The best LinuxWorld Expos were the early ones. There were two in 1999 alone, both in the San Jose Convention Center. The second one, in August, had an official attendance of 14,278. It felt like ten times that many. My favorite memory of that show was sitting among thousands of geeks packed into a vast space where they could hear (though barely see) Linus Torvalds speak, hanging on every word as if Linus were Billy Graham calling the Faithful to a crusade. Never mind that Linus' whole schtick was the antithesis of box office, and that most of what he wanted to talk about—as always—was incremental progress on the Linux kernel.

Linux energy back then was like the electric charge that swells in hills below a gathering thundercloud. The high-tension wires that crossed the computing world sparked and glowed with vast anticipation of a world where the advantages of open over closed, free over captive, common over exclusive, were all as plain as day to the Faithful—but to few others.

Now the storm has passed, lightning has flown, and the world we expected is largely here. In GhandiCon (**www.faqs.org/docs/jargon/G/GandhiCon.html**) terms (first they ignore you, then they laugh at you, then they fight you, then you win), we've pretty much arrived at GhandiCon Four. Of course, the future is not evenly distributed. Desktop Linux, for example, has been arriving asymptotically for years. All we need to close that gap is one smart hardware OEM move, which has to happen eventually. (See UpFront for the Dell IdeaStorm story, which is very encouraging.)

Meanwhile, the device drivers keep piling up. If we spelled out the whole LAMP stack, it would be more than 145,000 letters long. Today, it's kinda hard to build "solutions" to anything requiring computing and Net connections and not to take advantage of so many free and open building materials. There is also a huge demand market for smart techies who not only know how to build with those materials, but how to improve them as well.

Yet the number of Linux queries (**www.google.com/trends?q=linux**) on Google has trended downward during the past two years. Although the news volume has held steady, the query volume today is about half what it was at the end of 2003. (That's as far back as Google goes, and it doesn't give precise numbers.)

That's why fighting for Linux today is like fighting for geology, botany or the periodic table. There may be some holdouts around less sensible paradigms, but what's the point? The Linux Revolution has become the Linux Establishment. We've won. Now what?

Good question. (That's what you say when you don't know the answer. *Good question*.) Here at *Linux Journal*, we like a good cause as much as the next magazine. And, we'd like to celebrate Linux's victory in exactly the way you'd expect any born fighter to behave: by looking for new fights.

Fights are naturally interesting. That's what story theory says. For a story you need only three elements: 1) a *protagonist*—somebody or something you care about and can identify with; 2) a *problem* against which the protagonist struggles; and 3) *movement* toward a resolution. You don't have a story if your protagonist isn't interesting, the problem is pointless, or if there's no movement toward an end state. That's why sports and war stories are so compelling.

So, what will our story, or stories, be? I'll suggest four and leave the rest up to you.

Citizens vs. Carriers

Linux and the Net have grown together ever since Apache became the standard Web server in the mid-1990s. Yet while Linux rocks on, the Net is becoming trapped in carrier silos. Net users today are no less trapped by their phone or cable companies than personal computer users in 1999 were trapped by Microsoft Windows.

The difference is that every carrier is its own Microsoft, every Net service is as crippled as Windows, and customer choice (in the US, at least) is between Tweedle-telco and Tweedle-cableco—or just one of those. These carriers still look relatively good to customers because the connection speeds they offer (labeled "broadband" or "high speed") are many times higher than dial-up. It's too easy to forget that dial-up was what broke Net access wide open, making it available to nearly everybody—and did it in spite of the phone companies, rather than because of them. If it hadn't been for the original dial-up ISPs—The Little Garden, Panix, Batnet, Earthlink and even AOL—the Net still would belong only to universities, government and big business.

Now customers think the Net is gravy on top of their phone or cable TV services. They don't realize that the Net is the real base utility, and that it can carry any kind of gravy you like, including telephony and television. Almost nobody talks about all the businesses a wide-open Internet makes possible, mostly because the cablecos and telcos support consumption and discourage production. The "Net Neutrality" fight is a red herring. Most "high-speed Internet" customers have never experienced truly neutral service. Instead, they've enjoyed asymmetrical bandwidth and port blockages, without ever tasting what they've been missing.

Things are much better in some other parts of the world. Japan and Korea have notoriously high bandwidth at low prices, for example. The country with the highest broadband penetration is Denmark, with Estonia not far

behind. However, all is not rosy there either. Networks may be fast in Korea, but Microsoft's market share in many categories, including desktops, verges on 100%. Broadband growth in Europe has recently slowed in regions (including Denmark) where incumbent carriers are making comebacks.

The big fight here is between independence and dependence, between citizens and monopolies (or duopolies), between local initiatives—backed in many cases by local governments—and some of the nastiest state and federal politics you're ever going to find. In every case, the protagonists are individuals, local groups, local companies, local governments and local utilities.

The problem they face is a combination of duopoly entrenchment and well-lobbied protection at the federal and state levels. The telcos alone are the biggest-spending lobbying group in US history—even bigger than the pharmaceuticals. And, they don't just work Congress. Some carriers are working at the state level to make it against the law for anybody to carry the Internet other than themselves.

Linux folks can help enormously here, because Linux techies—our readers—know how to build good, strong, reliable, easily fixed and easily improved solutions. And, they know how to do it on the cheap. We've been watering grass roots for up to two decades or more. Stallman taught us what freedom means, and Torvalds taught us how to have fun putting it to use. We have a lot of leverage.

Mobile: the Ultimate User Space

Sometime this year there will be more than three billion mobile phones in the world. To put that in perspective (relying on last month's *LJ* Index), compare that to 1.4 billion credit cards, 1.3 billion land lines, 1.1 billion Net connections, 800 million cars, 200 million computer games, 100 million PVRs and 85 million iPods.

Cell phones are networked computing devices. A growing percentage of them run on Linux. Yet the OpenMoko (**openmoko.com**) and Trolltech's Qtopia Greenphone (**www.trolltech.com/products/qtopia/greenphone**), both wide-open working prototypes, are rarities. They face an enormous uphill battle against silo'd alliances between cell service carriers and equipment makers, such as Nokia and Motorola.

Yet the world needs open phones. In fact, I'd hazard a prophesy that open phones are inevitable, because there will be far more money to be made *because of* open phones than will ever be made *with* closed ones (and closed services offered only by carriers). We're starting to see vertical cracks in the closed wall of mobile telephony in settings such as universities, where rogue companies like Rave Wireless (disclosure: I consult them) provide students with custom (based on open) phones that run on familiar networks (such as Cingular and T-Mobile), but that do far more than the closed phones sold at stores by those same networks. Users are even free to do their own programming, create and add their own features and services. With each crack of this kind in a vertical market, the chance improves that open phones will become the norm rather than the exception.

The protagonists here are Linux techies, but working a much larger world of possibilities. The problem, as ever, is less a matter of closed systems than of the mentality behind it.

Once markets start to open up, it will be easy to fill whole magazine issues with stories of clever hacks and deployment successes.

Desktop and Laptop Linux

Desktop Linux has been approaching without ever arriving since the mid-1990s. Most *Linux Journal* readers are there already, of course. But they're wizards. The muggles are still on Windows and Mac boxes. What we've needed for the duration is one or more of the major hardware OEMs to wake up and smell the volume. Last year, Lenovo began selling Linux-loaded ThinkPads in a committed way, but not aggressively. Lenovo didn't push it. This year, Dell set out bait in the form of IdeaStorm, a site that had all the look of a "conversational" marketing ploy, but instead served as a hole in the Windows-only dike that has been holding the Linux desktop river outside of Dell's headquarters for the duration. That hole quickly widened to a river of its own, flooding through Dell's product development system. (See the IdeaStorm story in this issue's UpFront.)

I'd love to be a fly on the wall when Michael Dell tells Steve Ballmer that Dell will be selling Linux-branded laptops and desktops in a much more public way, because the company has no choice: the market demands it.

HP, Sony and the rest won't be far behind. As the volume grows, so will the portfolio of applications and the sum of expertise about Linux desktops and laptops.

This is a category that will explode very quickly. I'm willing to bet right now that in

Net users today are no less trapped by their phone or cable companies than personal computer users in 1999 were trapped by Microsoft Windows.

June 2008, *Linux Journal* will have an unavoidably personal focus to every issue—for the simple reason that there will be too much going on with desktops and laptops.

Or maybe not. We don't know yet. Lenovo, HP and Dell may continue quietly to fill orders for desktop Linux without ever marketing it aggressively. This won't go on forever, but the asymptote may still stay flat for another year, two or three.

Meanwhile, desktop and laptop Linux are still worth fighting for, just like we've been doing for the last decade or more.

DIY Everything

I have a confession to make. Or a *Make* to confess. I love *Make* magazine. I wish we'd done something like that first. Kudos to Dale Daugherty and the O'Reilly folks for pulling that one off and doing a great job with it—also for not running too much Linux-type stuff in there.

When I started with *Linux Journal* in the late 1990s, we were basically a how-to magazine. To a large degree, we still are. Most of our readers are hands-on types in any case. Problem solvers. Most of our writers (myself excluded) are too.

So I'm wondering...now that Linux is (or can be) in nearly everything, what can we make or fix that's one or more layers up? What can we do with MythTV that's beyond a set-top box? Pluto is a cool (and Linux-based) whole-home automation, security, entertainment and telecom system. But, it's still a system. A deep and underappreciated (and under-deployed) virtue of openness is modularity. You want to be able to mix and match different stuff from different makers, including (especially) yourself. We should be making Legos with Linux, not just embedding it in finished closed products that work only with themselves.

Here the fight is for the right and ability to build what you want, any way you want to build it. Although *Make* is oriented toward doing fun hacks on already-made stuff (turning a mouse into a robot or adding temperature control to a coffeemaker), we'd angle more toward making the modules, and the things-with-modules that allow anybody to build anything. Our protagonists would be the same DIY-ers we've had all along, but the problem would be Building Anything. Fun problem.

Years ago, I talked about how the software industry was turning into a construction industry—when architects, designers, builders and their specialties would all be independent of any one company's platform or development environment. Now we're almost there, but not quite. The fight here is to make Linux and its endless variety of "stacks" into the base materials with which people can put together using their own virtual Home Depots.

Freedom vs. Control

Although it's easy to point to the exemplary successes of Linux-built giants such as Google and Amazon, it's just as easy to overlook the degree to which the practical value system behind Linux development has become the default approach to networked progress.

Yet even as Linux and the LAMP+ stack have become standard building materials, there's nothing to stop them from being used in service of a proprietary mentality that seeks to lock in customers, lock out competition and lock down markets. As Steven Hodson puts it (**www.winextra.com/?p=354**):

> Many would like to believe that the best and strongest weapon against the old guard of technology is the Open Source movement, but what they don't see is that they have already been co-opted and have just become another way to make money. While the roots of the OSM (Open Source movement) may still technically be free to all, the old guard is quickly locking up parts of it with service contracts and corporate licensing.

It's still customary for VCs to ask their potential portfolio companies, "What's your lock-in?" This is an Industrial Age mentality that needs to be exposed as a value-subtracting anachronism in a world where creation and choice yield abundances that can be put to countless productive uses. You should want to build goods and provide services that customers choose freely. You should keep customers because they want to stay, not because you've trapped them in a silo.

Even Steve Jobs this year came out and said the record industry would be better off without DRM. That's because he's no less trapped than any of his customers.

The protagonist here is nothing less than the cause of freedom, which will never be old Gnus. (Pun intended.) The problem here—the enemy—is a mentality that's as old as the Industrial Age.

The battle for freedom, of course, is one we've been fighting all along. The difference now is that the logic of lockup is more and more exposed, and its flaws are more and more evident—though not yet widely obvious.

The fight, then, will shift from ideals to practical matters. How do you make money by building with free stuff and putting it to use, rather than just by selling it? How is software more useful and important as it becomes less and less of an industry? How do you get more work done, and become more valuable as a contributor because you're working with free and open goods?

These are still new questions, even though *Linux Journal* has been a living answer to all of them since 1994.

What's Your Story?

So now the question goes to the floor. What are the Good Fights you want to read about in *Linux Journal*? You tell us. Write to ljeditor@linuxjournal.com.■

Doc Searls is Senior Editor of *Linux Journal*. He is also a Visiting Scholar at the University of California at Santa Barbara and a Fellow with the Berkman Center for Internet and Society at Harvard University.

servers DIRECT™
MORE PRODUCTS, BETTER SERVICE, GUARANTEED.

GO STRAIGHT TO THE SOURCE!

SERVERS THAT WORK SMART

SAVE TIME AND REDUCE OPERATIONS COSTS WITH THE DUAL-CORE INTEL® XEON® PROCESSOR IN YOUR SERVERSDIRECT SYSTEM

1U Twin™ Innovation

* High Density Computing Technology
* Reducing Cost, Engery and Space Requirements
* Support up to **16 processor cores** Quad Xeon 5300 Series

SDR-6015T-TB 1U Data Center Clustering Server

Two systems (nodes) in a 1U form factor. Each node supports the following:

* Dual-processor Quad & Dual Core Intel® 64-bit Xeon® Support
* Up to 32GB DDR2 667 & 533 SDRAM Fully Buffered DIMM (FB-DIMM)
* 2x Intel® (ESB2/Gilgal) 82563EB Dual port Gigabit Ethernet Controller
* 2x Hot-swap SATA Drive Bays
* 900/980W High-efficiency Power Supply

SDR-2501T
2U Application Server

Ideal solution for a stable business-critical application server that minimizes deployment and support costs

* 2U Chenbro Rackmount Chassis with 600W power supply
* Supermicro X7DVL-E Server Board with Intel® 5000V (Blackford VS) Chipset
* Intel Dual-Core Xeon 5030 Processor 2.6GHz 667MHz
* Total 1024MB, 2pcs x 512MB Kingston DDR2 533MHz FB-DIMM ECC
* Seagate SATA II 80GB 7200 RPM 8MB Cache SATA 3.0Gb/s Hard Drive
* 6 x 1" Hot-swap SATA Drive Bays
* Intel® (ESB2/Gilgal) 82563EB Dual-port Gigabit Ethernet Controller
* Intel ESB2 SATA 3.0 Gbps Controller RAID 0, 1, 5, 10 support

STARTING PRICE **$1,299**

SDR-3111T
3u Mission Critical Application server

Designed to deliver exceptional availability, simplified manageability, outstanding performance and revolutionary scalability to help you build a cost-effective, flexible IT infrastructure

* 3U Chenbro Rackmount Chassis with 600W power supply
* Intel S5000PSLSATA Xeon 5000P Server Board
* Intel Dual-Core Xeon 5030 Processor 2.6GHz 657MHz
* Total 1024MB, 2pcs x 512MB Kingston DDR2 533MHz FB-DIMMECC
* Seagate SATA II 80GB 7200 RPM 8MB Cache SATA 3.0GB/s Hard Drive
* 12 x 1" Hot-swap SATA Drive Bays
* Dual-port Gigabit Ethernet Controller
* Intel SATA 3.0 Gbps 6-PORT Controller RAID 0, 1, 10 support

STARTING PRICE **$1,699**

SDP-7045B-TR+B

Ideal choice for small businesses looking for their first server or to upgrade an existing server

* Supermicro 4U Rackmountable / Tower Chassis with 800W High Efficiency Redundant Power Supply
* Supermicro Super X7DBE+ Server Board
* Intel Dual-Core Xeon 5030 Processor 2.6GHz 667MHz
* Total 1024MB, 2pcs x 512MB Kingston DDR2 533MHz FB-DIMM ECC
* Seagate SATA II 400GB 7200 RPM 16MB Cache SATA 3.0Gb/s Hard Drive
* 8 x 1" Hot-swap Drive Bays
* Intel® (ESB2/Gilgal) 82563EB Dual-port Gigabit Ethernet Controller
* Intel ESB2 SATA 3.0Gbps Controller RAID 0, 1, 5, 10 support

STARTING PRICE **$1,999**

SDR-5500T
5U advanced Storage Server

True SATA II 3Gb/s storage server configured with dual Xeon CPUs and room for a tape backup

* AIC 5U Rackmountable with 950W Triple Redundant Power Supply
* Supermicro Super X7DBE Server Board with Intel® 5000P (Blackford) Chipset
* Intel Dual-Core Xeon 5050 Processor 3.0GHz 667MHz
* Total 1024MB, 2pcs x 512MB Kingston DDR2 533MHz FB-DIMM ECC
* Seagate SATA II 400GB 7200 RPM 16MB Cache SATA 3.0Gb/s Hard Drive
* 24 x 1" Hot-swap Drive Bays
* Intel® (ESB2/Gilgal) 82563EB Dual-port Gigabit Ethernet Controller
* Intel ESB2 SATA 3.0Gbps Controller RAID 0, 1, 5, 10 support

STARTING PRICE **$5,399**

SERVERS DIRECT CAN HELP YOU CONFIGURE YOUR NEXT HIGH PERFORMANCE SERVER SYSTEM - CALL US TODAY!

Our flexible on-line products configurator allows you to source a custom solution, or call and our product experts are standing by to help you assemble systems that require a little extra. Servers Direct - your direct source for scalable, cost effective server solutions.

1.877.727.7887 | www.ServersDirect.com

intel Xeon® inside™
Dual-core. Do more.

Introversion Software's *DEFCON*

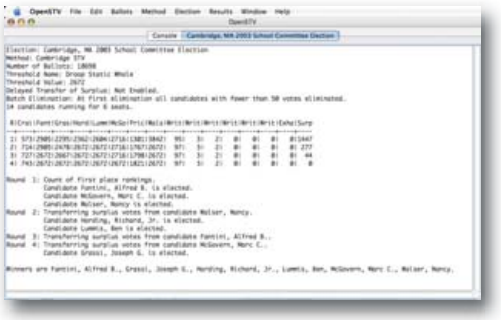
The UK's Introversion Software was proud to tell us that it is "keen supporters of the Linux community" and, therefore, is releasing its third and latest Linux-based game, *DEFCON*. *DEFCON* is an on-line, competitive, multiplayer strategy game based around the theme of global thermonuclear war. Inspired by the 1983 cult-classic *Wargames*, the game "evokes the tension, paranoia and suspicion surrounding the Cold War era". The player assumes the role of a general hidden in an underground bunker, whose mission is to exterminate the enemy's civilian population while simultaneously disabling the enemy's ability to retaliate. *PC Gamer UK* described *DEFCON* as "pure, deep, utterly unconscionable fun". A Windows version is already available. Introversion should get an award for best URL to boot!

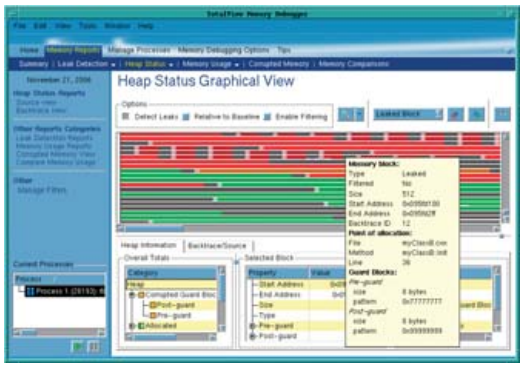
www.everybody-dies.com

OpenSTV

Ever feel like voting your conscience by supporting the Penguin Party rather than settling for the lesser of two "Republicrat" or "Demopublican" evils? To solve this dilemma, alternative (and Constitutional and increasingly popular) voting methods, such as single transferable vote (STV) and instant runoff voting have evolved that allow one to rank candidates in an election. If your Penguin Party candidate has no chance in hell to win, your vote counts instead for your lower-ranked choice who has a shot at winning. Sorting out these voting preferences is the job of OpenSTV, now in version 1.1, an open-source application that tabulates votes according to the respective voting rules. Data generally comes from from paper ballots and is dumped into OpenSTV. The lead developer says that "some of the voting rules have been extensively verified by comparing the results over hundreds of elections against other software". OpenSTV runs on Linux, Mac OS X or Windows and can be downloaded from SourceForge.

stv.sourceforge.net





TotalView Technologies' MemoryScape

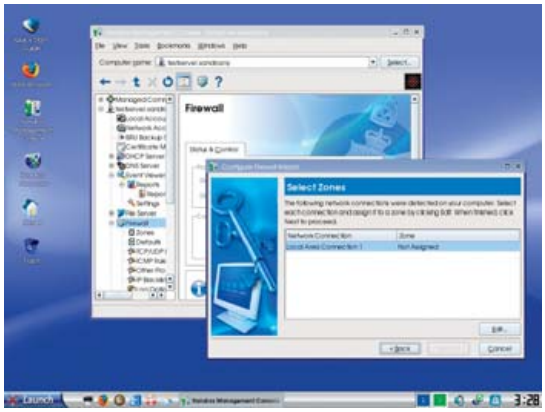
The company you've known as Etnus has rechristened itself as TotalView Technologies, and to celebrate, it has released version 2.0 of its MemoryScape standalone interactive memory debugger. MemoryScape "helps developers identify, inspect and resolve difficult memory problems in C, C++ and FORTRAN, including complex multiprocess and multithreaded programs", says TotalView. Some key features include tools that allow developers "to monitor heap memory, view memory usage, locate memory leaks, track memory events and show corrupted memory". Developers also can save and compare memory states, compile memory reports and find memory problems without recompiling. New features in MemoryScape 2.0 include support for MPI programs and remote memory debugging. A trial version is available for download from TotalView's Web site.

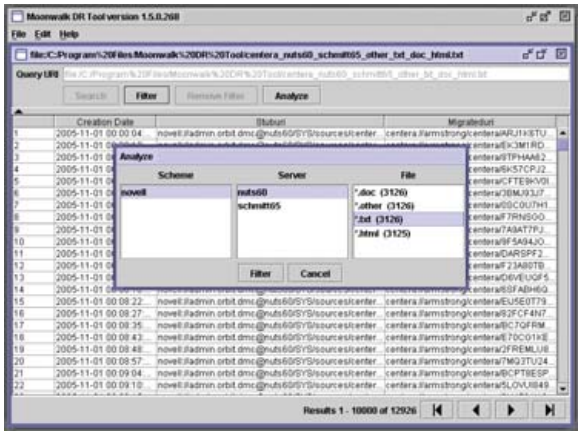
www.totalviewtech.com

Xandros Server

Xandros' new Server 2.0 just hit the streets and contains new features like integrated OpenDocument collaboration and comprehensive server backup and restore. The OpenDocument collaboration extension, created in tandem with the firm O3Spaces B.V., "provides OpenDocument and MS-Office document collaboration, management and retention services" and serves as an alternative to the Microsoft Office SharePoint server. For server backup and restore, Xandros has integrated SEP AG's "SEP sesam application, which provides comprehensive data security for the Xandros Linux Server, including full integration with its new Scalix 11 collaboration platform".

www.xandros.com





Moonwalk Software Suite

The Aussie firm Moonwalk made its own giant leap for our kind, this one to these North American shores, by unveiling version 6.0 of its self-titled, "all-inclusive data management and protection software". Moonwalk's raison d'être is to "automate and proactively manage the migration, copying and movement of data transparently throughout the enterprise" regardless of platform, including Linux, Windows, UNIX and NetWare. The application exploits secondary over primary storage by migrating, copying and moving data according to user-defined rules and policies based on criteria such as age, size, file type, filename, file creator and so on. It further "dispenses with tiered or hierarchical storage approaches and SRM applications that merely provide visibility into storage usage". Moonwalk is compatible with available backup solutions.

www.moonwalkinc.com

Concurrent Real-Time Computing Solutions

The real-time computing specialist, Concurrent, released three new products in April 2007, namely its RedHawk Linux 4.2, NightStar Tools 4.1 and SIMulation Workbench. First, the new release of RedHawk, Concurrent's real-time Linux OS, features a 2.6.18.8 Linux kernel with many of Ingo Molnar's accepted real-time patches, performance and stability enhancements, support for the latest Intel quad-core processors and 32/64-bit OSes on AMD Opteron processors. Second, NightStar Tools 4.1, an integrated toolset for developing time-critical applications, adds an enhanced Qt-based GUI, an application illumination feature and tuning enhancements. Finally, SIMulation Workbench is a new simulation software product to simplify real-time modeling, providing a complete framework to develop and execute real-time hardware-in-the-loop simulations.

www.ccur.com

Pogo Linux's StorageDirector 3000

Our pals at Pogo Linux passed on news of their new series of network attached storage (NAS) appliances, called StorageDirector 3000. The philosophy behind StorageDirector is to take a "simple, efficient approach to storage management", leveraging open architectures to reduce costs yet "still providing a high-end feature set" that targets Pogo's core customer, "the SMB with enterprise aspirations". Powered by the custom StorageDirector OS, the new product line enables the following: simple, secure management of storage and backup via a Web browser; cross-platform file sharing and utilization of all major file-sharing protocols; disaster recovery and backup; multi-pathing; advanced monitoring and alerts and both hardware and software RAID, including RAID 6 (double-parity). Customers can configure their own StorageDirector 3000 on Pogo's Web site.

www.pogolinux.com/go/sd3000

Woven Systems' EFX-1000 Ethernet Fabric Switch

Woven Systems has put more than a beach bucket's worth of VC money into its new switch product, the EFX-1000. The end result, says Woven, is the first of a new class of Ethernet Fabric Switches, intended to meet the needs that accompany multicore servers, server consolidation and virtualization, IP storage and data center grids. Ethernet Fabric Switches can be interconnected to build "resilient, low-latency, non-blocking meshed Layer 2 fabrics scaling to more than 4,000 10GbE ports". The 10GbE EFX-1000 switch "incorporates the performance and low cost of InfiniBand, the reliability of Fibre Channel, and the plug-and-play interoperability of Ethernet", all at a significantly reduced per-port price. Woven Systems has been dubbed one of the "Top 10 Startups to Watch" by the publication *Byte and Switch* due to its "potentially disruptive data center technology", as well as "Cool Vendor" by the Gartner Group.

www.wovensystems.com

Please send information about releases of Linux-related products to James Gray at newproducts@linuxjournal.com or New Products c/o *Linux Journal*, 1752 NW Market Street, #200, Seattle, WA 98107. Submissions are edited for length and content.

Open-Source Databases, Part III: Choosing a Database

Which database is right for you? MySQL or PostgreSQL?

REUVEN M. LERNER

If you are an application developer, you're probably working with large quantities of data. And, if that data is anything more complex than a hash table, you might want to consider moving some or all of it into a relational database. Relational databases are designed for reliable and flexible retrieval of data. The magic of a relational database is not the use of two-dimensional tables to store all of the information, but it's the fact that tables can be combined in many different ways and manipulated using the SQL query language.

As we saw in my database articles in the last two issues of *LJ*, open-source programmers are fortunate enough to have several database options at their disposal. By far, the two most popular open-source relational databases are MySQL and PostgreSQL. Each has a large and loyal following, and each continues to improve with every successive version.

And, when I write "large and loyal following" above, I'm not kidding. MySQL and PostgreSQL have long been at the center of a major flame war within the Open Source world. If someone on Slashdot dares say something about one of these products, you can be sure it won't be long before someone writes a nasty (and often childish) note about the other one. These disagreements often reflect the knee-jerk attitudes of uninformed users, but there have been no shortage of attacks from well-known and informed users of these products as well.

I believe there are circumstances when either MySQL or PostgreSQL might be an appropriate choice. I've strongly preferred PostgreSQL in my work during the last decade—yet, there definitely are times when MySQL seems to be the more appropriate solution.

So, despite my personal biases and the risk of opening a flame war within the Open Source community, I now conclude this series about open-source databases with a comparison between MySQL and PostgreSQL in a number of different categories. I hope by the time you finish reading this article, you understand that choosing a database is almost never a matter of finding the "fastest" or "best" product, because there is no one way to measure the quality or appropriateness of a relational database server. Rather, I hope you'll be able to consider each of these on the basis of its own merits, rather than on the propaganda that is so widespread.

Data Integrity

Perhaps the first and foremost task of a database is to store and retrieve data reliably. Just as you wouldn't want to use a hard disk that occasionally loses data, you don't want to put things into a database that occasionally mangles its contents. This is true even if the reliability comes at the expense of speed.

The gold standard for reliability in the database world has an acronym, ACID (Atomicity, Consistency, Isolation and Durability). This means that under all circumstances in the database, the following hold true:

- Atomicity: each query is guaranteed to complete or not, without any possibility of halfway or incomplete states.
- Consistency: the database is always in a legal state before and after a transaction.
- Isolation: each transaction occurs separately from other actions, so that you can't have two transactions interfering with one another.
- Durability: transactions persist over time, typically by being stored on a filesystem.

The attitude toward ACID within the PostgreSQL community has been unchanged since I first started to use it a decade ago, placing it as the highest possible priority. This doesn't mean PostgreSQL is lacking in other features, but rather it means the developers have worked to ensure that data stored in a PostgreSQL system will be consistent and reliable, even if you do nasty things such as issue a `kill -9` or pull the plug.

During the past few years, PostgreSQL has begun to offer even better support for transactions and database stability, using write-ahead logs (WALs) that describe each action taken by the database. These WAL files can be used to recover from a disaster or even to recover the database to an earlier point in its history—a feature known as point-in-time recovery (PITR). Thus, if you know something happened yesterday, but the database was working perfectly two days ago, you could use PITR to recover to the earlier, stable state. Recent versions of PostgreSQL also support two-phased commit, a type of transaction you're likely to see in a distributed system where multiple servers must coordinate their actions.

MySQL has had a mixed attitude toward ACID during the years. When I first started to use MySQL in 1995, the authors' attitude was that transactions should be handled by the application, not the database. Indeed, as recently as 2000, the to-do list for MySQL included tasks having to do with production-quality transaction-safe tables. This has led to a great deal of bad blood between the MySQL and PostgreSQL communities, with members of the latter sometimes claiming that no critical data should ever be stored in MySQL.

The good news is that modern versions of MySQL do indeed support transaction-safe tables, using InnoDB, a third-party product released under the GPL that has been integrated into MySQL for several years. Moreover, InnoDB appears to use techniques that PostgreSQL and Oracle have used for years, such as MVCC (multi-version concurrency control). The bad news is that at least some benchmarks I've

seen indicate that InnoDB has some problems scaling to large numbers of simultaneous queries.

In addition, the company that develops InnoDB recently was bought by Oracle, which might lead some people to worry about future licensing, development and pricing issues. For the time being, this latter issue does not appear to be a serious one, because Oracle and MySQL signed a contract in 2006 extending the licensing for InnoDB. But, MySQL does not appear to be taking any chances and has hired several experts to create a new table structure that will be owned by MySQL and thus be impervious to such business problems.

I'm personally of the persuasion that true ACID compliance is always a good thing to have around, much like seat belts in a car. Sure, you can drive without a seat belt, and the odds are that nothing will happen to you. But, it's impossible to predict when something bad might happen, and you really don't want to be without a seat belt under such circumstances. In the same way, if your data is important to you, it's best to ensure that it will persist with integrity.

A related problem has to do with the degree to which each database enforces constraints and limits. PostgreSQL tends to be quite stringent on such matters, refusing to accept illegal data. MySQL tries to be more forgiving and flexible, but that can result in strange and illegal data being stored.

For example, consider the following set of MySQL commands, in which we create a table foo with a single column (named a) of type DATE:

```
mysql> CREATE TABLE foo (a date);
Query OK, 0 rows affected (0.08 sec)

mysql> INSERT INTO foo (a) VALUES ('2007-feb-30');
Query OK, 1 row affected, 1 warning (0.00 sec)

mysql> SELECT * FROM foo;
+------------+
| a          |
+------------+
| 0000-00-00 |
+------------+
1 row in set (0.00 sec)
```

By contrast, this is what happens in PostgreSQL:

```
atf=# CREATE TABLE foo (a date);
CREATE TABLE

atf=# \d foo
Table "public.foo"
 Column | Type | Modifiers
--------+------+-----------
 a      | date |

atf=# INSERT INTO foo (a) VALUES ('2007-feb-30');
ERROR:  date/time field value out of range: "2007-feb-30"
```

It is possible to configure MySQL to be more strict on such issues, but most users will not think to do so and will be stuck with illegal values in their tables.

Given the political and technical issues at MySQL, as well as the weird (and potentially dangerous) default behavior in MySQL, I believe that PostgreSQL has a big edge on issues of data integrity.

Features

MySQL and PostgreSQL offer a very large number of built-in features, many of which have been added in response to community requests and reactions. Both offer a large number of data types, which can be mixed and matched within a single row without restrictions. (The very limiting row-length restriction that plagued versions of PostgreSQL is now ancient history, I'm happy to say.) Both databases now support Unicode characters; MySQL supports both UCS-2 and UTF-8 encoding, and PostgreSQL supports only the latter.

Both databases also offer a very large number of functions that manipulate data, including strings and dates. It's quite convenient to be able to compare and sort dates or to find all rows whose timestamp was within the last 24 hours. PostgreSQL's interval data type, which describes a length of time (rather than a particular point in time), has proven to be particularly useful. MySQL has a number of different types that database purists like myself frown upon, such as SET and ENUM, but that are undoubtedly popular for many users.

In many areas where PostgreSQL has had an advantage, MySQL is beginning to catch up. PostgreSQL users have long been able to create new data types and functions that operate on those types. Indeed, PostgreSQL offers developers the unusual ability to write server-side functions in a number of languages, including SQL, Perl, Python, Java, Tcl and the R statistical language. MySQL does not allow for the creation of new data types, but recent versions do provide the ability to write server-side functions and stored procedures.

MySQL has offered a built-in solution for full-text search, accomplished by using a special type of index on text fields. However, there are some important restrictions on this index, such as the fact that it works only with MyISAM tables. Given that these tables support neither foreign keys nor transactions, I am a bit nervous about suggesting them as a solution.

PostgreSQL's full-text search solution (tsearch2) has the opposite problem. Although it is robust and works well within PostgreSQL's standard transactional tables, it requires some work to configure and install. Most administrators and programmers will be able to install it successfully within a short period of time, but nonetheless, there is a difference between a built-in capability and one that needs to be added.

PostgreSQL has a number of built-in features that MySQL either has yet to implement or that are scheduled for future releases. Among these are the ability to use subselects anywhere in a query, the use of sequences (rather than simple auto-increment columns), rules that allow users to modify the way queries are interpreted on a given table and CHECK constraints on column values. Recent versions of MySQL now include features that were previously available only in PostgreSQL, such as triggers and views.

In general, the PostgreSQL development group seems to emphasize SQL standards more than MySQL does, although the MySQL developers appear to be increasingly sensitive to this need and now offer an --ansi command-line switch for those people who want to work in a standards-compatible mode all of the time.

Both MySQL and PostgreSQL are extremely easy to use. Each

comes with a command-line client program that is packed with features, allowing you to manipulate your database by sending SQL queries. I have become spoiled by some of the features of the PostgreSQL command line, such as the expanded output (\x).

The command-line interfaces for both databases have grown more useful over time. Although the MySQL interface might appear to have fewer commands, that's partly because MySQL has made some data available via SQL queries (for example, SHOW TABLES), which would require more complicated queries in PostgreSQL, leading to the creation of a shorthand command, \dt. Both command-line interfaces use GNU readline, making it easy to edit and re-issue queries. Both also allow users to edit the previous query using the \e command.

Overall, it's probably fair to say that PostgreSQL offers a superset of MySQL's capabilities, aside from a few issues (for example, built-in text indexing). Those capabilities that PostgreSQL does not have, such as new data types and functions, are added into the system easily, without needing to recompile or otherwise modify the core PostgreSQL server. That said, I believe MySQL's capabilities are nothing to sneeze at and are likely more than adequate for most applications you might be writing.

Administration

Both MySQL and PostgreSQL are amazingly easy to administer, especially in small- and medium-size cases. You (optionally) change a few configuration options, start the server and then walk away. There's really not much more to do than that. For anyone who has worked with a larger database system, such as Oracle, this is a refreshing change. However, there are slight differences in the ways the two systems operate.

PostgreSQL relies on several external UNIX-level commands to create and manage databases and users, as well as the activity of the PostgreSQL server. There is no central PostgreSQL administrative program. MySQL, by contrast, has a central mysqladmin program that handles most functions having to do with server startup and shutdown, as well as the creation and destruction of databases. The creation and management of users is handled by manipulating tables in the mysql database.

PostgreSQL's counterparts to the mysql database are special system tables and views, all of which begin with the pg_ prefix. These tables, although necessary for the system to run, easily can be ignored by most programmers and come into play only when trying to tune the system or figure out how to optimize queries.

GUI-based administration tools are available for both programs, as well as Web-based tools written in PHP. To be honest, I haven't used these tools much during the years, given my familiarity with (and preference for) command-line systems for working with databases. However, my experience with both sets of GUI programs has been positive, and my impression is that they are both stable and secure, as well as useful.

Another aspect of administration unique to PostgreSQL is the need to "vacuum" dead rows from the database to return them to the operating system or to other rows that could benefit from the space. In addition, PostgreSQL's vacuum function visits the rows of the dead and uses the statistics it collects to inform the optimizer and query planner. Nowadays, the auto-vacuum dæmon takes care of this automatically for most people, removing the long-dreaded need to schedule it in cron.

One administrative area that is particularly hot right now is replication. Many Web sites and other applications are pushing the limits of their database servers, and it would be useful to split the work among multiple servers. Of course, this raises issues of data integrity and synchronization among distributed processes. The simple solution to the problem is to have a master/slave relationship among the different servers, with UPDATEs and INSERTs taking place only on the master server, and SELECTs taking place on the slave servers. Solutions for this exist under both MySQL and PostgreSQL, although the PostgreSQL solution (Slony) is external to the standard package and apparently can be difficult to install and configure.

A more complicated setup involves the use of two master database servers. MySQL appears to have taken the lead on this front with a relatively new clustering tool. But, PostgreSQL users, who have been clamoring for such tools for several years now, appear to be on the verge of getting their wishes fulfilled.

Finally, no database server would be worthwhile if it weren't possible to perform regular backups. pg_dump and mysqldump are command-line programs that turn the current contents of a database into a text file. Such dump files are quite useful and can be used to rebuild the database when necessary.

I would argue that when it comes to administration, the two database products are identical—unless you need replication, in which case you'll probably benefit from MySQL's greater experience and replication integration.

Performance

For years, one of the claims made in the MySQL/PostgreSQL flame war has had to do with speed. MySQL fans often have claimed that their system is faster, particularly for read-only tasks, making it a superior choice for Web sites where most data is read. PostgreSQL advocates, in contrast, claim that their system holds up to big loads much better than MySQL.

I haven't conducted any benchmarks of my own, but my reluctance to do so is an admission that I'm unqualified to create a good

Resources

The PostgreSQL home page is **www.postgresql.org**. Similarly, the MySQL home page is **www.mysql.org**. Each has its latest manuals posted, as well as software, drivers and discussion lists.

A table comparing administration and programming of the two databases is available at **linuxboxadmin.com/articles/postgresql-for-mysql-users.php**.

A relatively recent comparison of the two databases' performance is at **www.mysqlperformanceblog.com/2006/11/30/interesting-mysql-and-postgresql-benchmarks**, which points to the following: **tweakers.net/reviews/657**.

Finally, a comparison between the databases (but perhaps a bit out of date), along with Oracle, was conducted at CERN, the European center for particle physics, and is available at **dcdbappl1.cern.ch:8080/dcdb/archive/ttraczyk/db_compare/db_compare.html**.

benchmark, and not that I believe the two systems are identical or that performance isn't important. Moreover, as I stated previously, I believe that performance is secondary to data integrity. I would much rather have a slow, reliable database than a fast one that occasionally will wreak havoc on my data.

From the benchmarks I've seen, it appears that MySQL is indeed faster than PostgreSQL when working with a small number of clients or with read-only data. However, all of the comparisons I've seen over the last few years indicate that as more clients are added to the system, PostgreSQL handles the load better.

Does this mean that PostgreSQL always will be faster? Of course not. But, it does mean that on particularly popular sites, PostgreSQL may hold up better.

Maybe I'm simply naïve, but I decided several years ago that I would largely ignore the performance debate when it came to databases. Both MySQL and PostgreSQL have large followings and have been used on large-scale systems. The data seems to indicate that PostgreSQL has an advantage, but enough people are using MySQL on large Web sites that I have to assume it is working well enough for them.

Support

Finally, no comparison would be complete without mentioning support. We might consider several types of support—from the strength of the Open Source community to the number and quality of companies supporting (and developing) the software to the number of third-party applications that support each database.

It is impossible to ignore the extremely large number of MySQL users in the world. This has led to an outpouring of books, tutorials and mailing lists for MySQL—some (but not all) of which have been sponsored by the MySQL company itself. If the community-based support is not enough, it is possible to buy commercial support for MySQL from a number of companies, including MySQL AB.

PostgreSQL has a smaller community, and a smaller number of books and tutorials available. However, my experience has been that the community is responsive to questions and suggestions, and that the lead developers often are quite willing to answer questions from all levels of users.

Many open-source packages support both MySQL and PostgreSQL. But, it is rare to find a package that supports PostgreSQL exclusively, and it is easy to find packages that support MySQL alone. This has been a source of some frustration for members of the PostgreSQL community; however, there doesn't seem to be much anyone can do about it, short of asking for patches or contributing such patches.

A recent thread on the main PostgreSQL mailing list asked about CRM packages that support the database. Although there were a few, there was definitely some grumbling about the lack of PostgreSQL from other open-source projects. Those projects often are staffed by small groups of volunteers who rarely understand how they can make their SQL more portable and thus easier to use on multiple brands of databases.

The bottom line on support is that although PostgreSQL support is excellent, MySQL support is overwhelming. If there is a winner here, it's MySQL.

Conclusions

So, should you pick MySQL or PostgreSQL for your next database task? All things being equal, I strongly recommend PostgreSQL. Its community might be smaller, and there are fewer resources available in print and on the Web. But, it has more features to ensure data integrity, its features are largely a superset of MySQL, and it always offers transactions and referential integrity, without having to specify a particular type of table.

That said, there are reasons to use MySQL: if you already are using it, if you need commercial or community support, if you need replication, or if you are using software that is incompatible with PostgreSQL, MySQL is a fine choice. Just make sure to use InnoDB tables, so that you can take advantage of what a database always was meant to do—ensure the quality of the data.■

Reuven M. Lerner, a longtime Web/database consultant, is a PhD candidate in Learning Sciences at Northwestern University in Evanston, Illinois. He currently lives with his wife and three children in Skokie, Illinois. You can read his Weblog at altneuland.lerner.co.il.

INTERVIEW WITH SIMON PHIPPS

Simon Phipps defends the open-source roots of Sun and the GPL-ization of Java.

Glyn Moody

Before joining Sun in 2001, Simon Phipps spent ten years at IBM, where he was Chief Java and XML Evangelist. He first came across free software in the late 1980s, when he was selling freeware from home as a sideline while working at Unisys. Today, Phipps is Sun's Chief Open Source Officer, and he plays a key role as the company moves its entire software portfolio to open source.

GM: What was the state of the open-source activity at Sun when you joined?

SP: I regard Sun as the original open-source startup company. It's the first long-lived company I can think of that used open source as the basis of a business model. If you look back down Sun's history, through the decades, you see that Sun kept on working openly with communities around software and hardware. It did it with NFS, it did with TCP/IP, it did with Java, and it's continuing to do it now with OpenSolaris and the rest of the portfolio. So, in one sense, there was always and there remains a very, very strong open software ethos at Sun.

When I joined Sun, Sun was right at the forefront of some really quite radical commercial open-source experiments—one of them, starting OpenOffice.org, and the other, getting NetBeans started. Inside Sun, people were talking strongly about the open-source future for Solaris because its heritage had been open source, and it was pretty clear that it was a good thing for its future to be open source as well. So, it was 2000/2001 that the legal clearance work for Solaris started off.

GM: What did that involve?

SP: The process you have to go through when you've got a piece of 20-year-old code is working out who owns it all. It's actually a reasonably intractable problem, because over the decades, the standard of proof has gradually gotten stronger and stronger. Just because there's nothing in the file header, doesn't mean it doesn't belong to someone else. Someone else might have the right to it. So we had an absolutely stunning team of people in the Solaris group who did what you might call licensing archaeology—looking at the code, comparing it with other code whose provenance was known, looking at variable naming styles and indentation styles, the language of the comments, trying to get a reasonable standard of proof about where the code had come from.

There are things that were much more obvious as to where they'd originated, and we were able to look back in our legal archives at the licenses and work out whether we had the rights. And, for quite a lot of those we then had to go back and negotiate third-party sublicensability. In many cases, we found that people were perfectly happy just to give us that right. There were some cases where we found the licensee didn't exist any more; finding out who abandoned code belongs to is another art form.

Something else you have to look out for is code that was based on disclosure of information where the disclosure itself was under trade secret terms. So, although we'd written the code, we didn't actually have the right to disclose it, because to do so would have disclosed the trade secrets. And, we still have problems with some of that code, in particular from video card manufacturers, where they were willing to tell us under trade secret terms how their chipset works, but they continue to refuse to allow us actually to disclose the source code.

GM: Presumably parallel to what you were thinking about licenses—how did you end up with the CDDL license?

SP: Solaris flowed out of BSD. It was actually BSD merged with System V at one point, so there were a lot of people who felt that we should be using BSD as a license. However, a lot of the innovations that have happened in Solaris have been by quite recently hired people—younger, brilliant engineers. And, there was a strong view as well that we should be using the GPL. Also, there were a lot of people who felt that the Mozilla approach to

licensing had been correct.

The difficulty with using the GPL was that it became clear as we did the licensing archaeology that there were going to be places in the software where we couldn't negotiate a free license for the source code. And, there were going to be some quite important places in the code, and there were going to be quite a lot of them for a long time. And, that was probably the deciding factor not to use GPL but to use the Mozilla license.

We looked at the Mozilla license, and we realised that we couldn't use it as it was. We saw that license proliferation was an increasing problem. So, we decided that we would have a go doing a good thing for the Free Software and Open Source community, by making the last clone of the Mozilla license that would ever need to be made, by parametrising it. We left as much of the language as identical as we could and produced the Common Development and Distribution License, the CDDL. The CDDL is, in my view, an absolutely excellent license; if anyone but Sun had written it, it would have been hailed as brilliant.

GM: How do you see the relationship between OpenSolaris and GNU/Linux?

SP: I think that there is a huge overlap in those worlds. One of the interesting things you discover when you run an OpenSolaris distribution like Nexenta is, my goodness, it looks just like Ubuntu. And you know, there's a really good reason for that: because it is Ubuntu, but it's got a Solaris kernel in it.

When you look at what UNIX-like operating systems really are, each is a set of editorial choices about which free software userland to assemble around which kernel. So you discover that everyone in Fedora, and everyone in FreeBSD, and everyone in OpenSolaris are all using the same stuff. They're all using GNOME or KDE, they're all using Sendmail, and they're all using Mozilla.

In the Solaris community, they've tended to use the OpenSolaris ON, which is the OpenSolaris jargon for the kernel and network. ON stands for Operating System/Networking. And the userland that's around it, well, there is a style of userland that's used by people who run servers. And then there's the style of userland used by developers, and that's typically GNOME or KDE, and tools. And it's just the same on Solaris as it is on Fedora. The look and feel is different, the editorial choices about where to put the icons are different, but ultimately, it's all the same stuff.

So I think we're going to see a gradual shift in the way we think about UNIX-like operating systems as we go forward. We are going to see much less of people trying to arbitrarily distinguish between the different peers in that community. I think that the distinction people try to force between UNIX and Linux is part of a strategy by corporations to diminish their competitors' versions of UNIX. And I think it harms us all, because the real competitor out there isn't somebody else's UNIX-like operating system, it's actually the closed stuff. I think that in the future, we'll see people who are deciding to run a Debian userland with a Solaris kernel, as well as people who are using a Solaris-inspired minimalist install with a Linux 2.6 kernel. We'll see people starting to use the BSD kernel together with KDE and a package manager that they got from the Solaris community.

GM: What's the history behind the opening up of Java?

SP: What was happening in the Open Source world [in the late 1990s] was the realisation that with-source-style licensing was commercially viable. We saw Mozilla being released, and then we saw the Open Source Initiative picking up the Debian Social Contract and turning it into the Open Source Definition. Meanwhile, over at Sun, everything was incredibly busy; there was way more business than Sun could cope with. It really didn't have the bandwidth to cope with this stuff that was happening over in the Open Source community. So it largely ignored it because the days were already full.

What's more, the people who were doing the open-source stuff were really pretty hostile to Java. Their hostility wasn't moderated by a recognition that Java came from what would now be recognised as an open-source company. The Open Source movement is busily accepting grace from IBM to promote Linux—shall we say, not entirely in isolation from the fact that Linux isn't Solaris. And, an engine of bad feeling was busy humming away nicely, between 2000 and 2003, with all the players doing their utmost to make sure that understanding and cooperation didn't break out.

GM: So what happened?

SP: Sun had a sort of near-death experience, when the [dot-com] bubble burst. It saw its stock price go down to a tenth of its previous value, and it saw the need to dismiss large numbers of staff. It became suddenly very obvious that lots of the people who didn't share Sun's values didn't belong here anymore; it also became very obvious that some of the approaches to software that Sun had been taking weren't actually in keeping with Sun's long-term values.

What then changed around about 2003, was it became obvious that Java had a huge international community. It actually had a huge Open Source community, developing on top of the Java platform, using open-source tools like Spring, Hibernate, JBoss and so on. And, I think it gradually became more and more obvious to people that this community probably was no longer as vulnerable to monopolisation as it had been. And, that meant it was less and less important to keep as the number-one priority the prevention of monopolisation, and it became acceptable to begin to think about other priorities for the licensing of the Java platform.

GM: What about the license? How did you end up making the surprising choice of the GPL?

SP: We looked at pretty much every license you can imagine for the Java platform. Obviously, lots of people thought we were going to use CDDL. There were several questions we had to ask ourselves: one of them was which license was most likely to prevent monopolisation by somebody loving us to death. Another factor was asking who wasn't using Java. And, Java is actually pretty widespread. It's on five billion devices, it's on eight out of ten cell phones, and it's used on a strong majority of enterprise application servers.

So the question has to be, well, how can you grow when you've already got such a strong market? And, the obvious place that we could grow was actually to GNU/Linux. If you look at the use of GNU/Linux outside Europe and North America, you discover that distributions like Fedora and Debian are actually very, very important in those geographies. And, none of those distributions were actually carrying Java, because of the licensing concerns.

Worse than that, it had been such a long time since they'd carried Java, that their package management systems had grown up in such a way that the versions of Java that Sun was actually making for those platforms didn't install. So, there was no way you could apt-get Java on Debian, for example. All Sun made available was an RPM. And, yes, if you were resolute, you could force it in there, but fundamentally, Java just wasn't available for Debian.

We felt that the biggest impact we could have on the Java market was by settling the long dispute with the GNU/Linux community. We felt that doing that would grow the market to everyone's benefit.

As an application developer on GNU/Linux, you don't want to have to worry about which version of the kernel is in use and which desktop environment people have, and which package management system they're using; you don't want to have 500 different versions of your installer for all the different versions of GNU/Linux out there. So, Java's got a really

strong value that it can offer the GNU/Linux community allowing not every application, but a lot of applications that are not too tightly coupled, to the system internals to be written using a platform-independent programming mechanism. And, that's the chief value that Perl and Python and others were bringing to the platform. And we thought, well, it's a great fit.

Another driver in choosing the GPL was we felt that the behaviour that would lead to monopolistic abuse of the Java platform would typically be done in secret until it was launched. By using the GPL, people will find it very tricky to do extensive development in secret. So we felt that the GPL provided us with the strongest protection against misbehaviour by monopolists in the Java platform.

GM: In a speech a couple of years back, Sun's CEO Jonathan Schwartz argued that the GPL is "IP colonialism", because he claimed it imposed on poorer countries "a rather predatory obligation to [give back] all their IP to the wealthiest nation in the world". Why did he change his mind?

SP: Well, you know, this is an interesting thing to contemplate, because I'm not sure he was wrong. The GPL does require you to set aside commercial protections for your software. And, it is possible that the use of the GPL for the indigenous software industry, for example, in Brazil, might harm the Brazilian economy. If you actually read the argument that Jonathan was making at the time, it's a good academic argument. What made it controversial was that it was the Chief Operating Officer of Sun saying it.

GM: To what extent is Sun now committed to opening up everything that it can?

SP: We're completely committed to open-sourcing every thing that we're able to. Jonathan Schwartz asserted that as our position about two years ago. We're well on the road to fulfilling that commitment.

GM: Given this commitment to free software, and the Open Source community, wouldn't it be more sensible for Sun to join Eclipse rather than pushing NetBeans on its own?

SP: The deal here is once again how you view open source. Open source isn't something you join; it's something you do. And, Sun doesn't want to join the Eclipse community, just like IBM doesn't want to join the OpenOffice.org community. Why would we want to join a community that isn't making any code we want to use in a product?

[Eclipse] has significant disadvantages for Sun in the way that IBM decided to set it up. If you want to join the board, you have to pay out a very large sum of money, you have to commit a certain number of engineers to work only on that, at the direction of the Eclipse community, and you have to guarantee to produce products that use the Eclipse core code within a year. And, for us to fulfill those requirements, we would need to drop NetBeans. And, NetBeans is a big part of our tools development.

GM: Finally, what's your vision of the world where open source is a major part of computing?

SP: I actually think that we're in the middle of a pivot point in the way society functions. I believe the World Wide Web as the vehicle for popularising the Internet is producing something that is as impactful as the Industrial Revolution. And, I think during the next decade, we will see that process of changing how absolutely everything works rolling out in front of us.

Pre-World Wide Web, most things that happened in the world were done on a hub-and-spoke basis where you'd have, for example, government in the middle and citizens on the end of the spokes. Or, you'd have industry in the middle and customers on the end of the spokes. I think the introduction of the World Wide Web has changed the basic topology of society from hub-and-spoke to mesh.

Because the software industry is so closely connected to the World Wide Web, it's been one of the first to be impacted. So I see open source as an inevitable consequence of the switch to a meshed world. It's, in my view, the dominant way that software is developed in a participation age. The way you make money is not by locking people in with a license at the beginning, but rather by providing the capabilities people want once they're running things.

In the meshed world, what helps you be successful in a business is influence. And, you get influence not by power but by being valuable. My vision is that we're switching over to this new world of influence instead of control, of value instead of power, of participation instead of distribution.

So, we can expect to see a rolling tide of change where the principles of the meshed society begin to be worked out in other areas, like politics, like journalism, like the way families function, like the way money is handled, represented and stored. All of these things will gradually fall under the influence of the meshed society. And, I think being at the forefront of working with that meshed society is going to serve Sun and the people in the Open Source communities very well.■

Glyn Moody writes about free software and open source at opendotdotdot.blogspot.com.

Programming Python, Part I

This tutorial jumps right in to the power of Python without dragging you through basic programming.

Python is a programming language that is highly regarded for its simplicity and ease of use. It often is recommended to programming newcomers as a good starting point. Python also is a program that interprets programs written in Python. There are other implementations of Python, such as Jython (in Java), CLPython (Common Lisp), IronPython (.NET) and possibly more. Here, we use only Python.

JOSÉ P. E. "PUPENO" FERNÁNDEZ

Installing Python

Installing Python and getting it running is the first step. These days, it should be very easy. If you are running Gentoo GNU/Linux, you already have Python 2.4 installed. The packaging system for Gentoo, Portage, is written in Python. If you don't have it, your installation is broken.

If you are running Debian GNU/Linux, Ubuntu, Kubuntu or MEPIS, simply run the following (or log in as root and leave out sudo):

```
sudo apt-get install python
```

One catch is that Debian's stable Python is 2.3, while for the rest of the distributions, you are likely to find 2.4. They are not very different, and most code will run on both versions. The main differences I have encountered are in the API of some library classes, new features added to 2.4 and some internals, which shouldn't concern us here.

If you are running some other distribution, it is very likely that Python is prepackaged for it. Use the usual resources and tools you use for other packages to find the Python package.

If all that fails, you need to do a manual installation. It is not difficult, but be aware that it is easy to break your system unless you follow this simple guideline: install Python into a well-isolated place, I like /opt/python/2.4.3, or whatever version it is.

To perform the installation, download Python, unpack it, and run the following commands:

```
./configure --prefix=/opt/python2.4/
make
make install
```

This task is well documented on Python's README, which is included in the downloaded tarball; take a look at it for further details. The only missing task here is adding Python to your path. Alternatively, you can run it directly by calling it with its path, which I recommend for initial exploration.

First Steps

Now that we have Python running, let's jump right in to programming and examine the language as we go along. To start, let's build a blog engine. By engine, I mean that it won't have any kind of interface, such as a Web interface, but it's a good exercise anyway.

Python comes with an REPL—a nice invention courtesy of the Lisp community. REPL stands for Read Eval Print Loop, and it means there's a program that can read expressions and statements, evaluate them, print the result and wait for more. Let's run the REPL (adjust your path according to where you installed Python in the previous section):

```
$ python
Python 2.4.3 (#1, Sep  1 2006, 18:35:05)
[GCC 4.1.1 (Gentoo 4.1.1)] on linux2
Type "help", "copyright", "credits" or "license" for
more information.
>>>
```

Those three greater-than signs (>>>) are the Python prompt where you write statements and expressions. To quit Python, press Ctrl-D.

Let's type some simple expressions:

```
>>> 5
5
```

The value of 5 is, well, 5.

```
>>> 10 + 4
14
```

That's more interesting, isn't it?

There are other kinds of expressions, such as a string:

```
>>> "Hello"
'Hello'
```

Quotes are used to create strings. Single or double quotes are treated essentially the same. In fact, you can see that I used double quotes, and Python showed the strings in single quotes.

Another kind of expression is a list:

```
>>> [1,3,2]
[1, 3, 2]
```

Square brackets are used to create lists in which items are separated by commas. And, as we can add numbers, we can add—actually concatenate—lists:

```
>>> [1,3,2] + [11,3,2]
[1, 3, 2, 11, 3, 2]
```

By now, you might be getting bored. Let's switch to something more exciting—a blog. A blog is a sequence of posts, and a Python list is a good way to represent a blog, with posts as strings. In the REPL, we can build a simple blog like this:

```
>>> ["My first post", "Python is cool"]
['My first post', 'Python is cool']
>>>
```

That's a list of strings. You can make lists of whatever you want, including a list of lists. So far, all our expressions are evaluated, shown and lost. We have no way to recall our blog to add more items or to show them in a browser. Assignment comes to the rescue:

```
>>> blog = ["My first post", "Python is cool"]
>>>
```

Now blog, a so-called variable, contains the list. Unlike in the previous example, nothing was printed this time, because it is an assignment. Assignments are statements, and statements don't have a return value. Simply evaluating the variable shows us the content:

```
>>> blog
['My first post', 'Python is cool']
```

Accessing our blog is easy. We simply identify each post by number:

```
>>> blog[0]
'My first post'
>>> blog[1]
'Python is cool'
```

Be aware that Python starts counting at 0.

Encapsulating Behavior

A blog is not a blog if we can't add new posts, so let's do that:

```
>>> blog = blog + ["A new post."]
>>> blog
['My first post', 'Python is cool', 'A new post.']
```

Here we set blog to a new value, which is the old blog, and a new post. Remembering all that merely to add a new post is not pleasant though, so we can encapsulate it in what is called a function:

```
>>> def add_post(blog, new_post):
...     return blog + [new_post]
...
>>>
```

def is the keyword used to define a new function or method (more on functions in structured or functional programming and methods in object-oriented programming later in this article). What follows is the name of the function. Inside the parentheses, we have the formal parameters. Those are like variables that will be defined by the caller of the function. After the colon, the prompt has changed from >>> to ... to show that we are inside a definition. The function is composed of all those lines with a level of indentation below the level of the def line.

So, where other programming languages use curly braces or begin/end keywords, Python uses indentation. The idea is that if you are a good programmer, you'd indent it anyway, so we'll use that indentation and make you a good programmer at the same time. Indeed, it's a controversial issue; I didn't like it at first, but I learned to live with it.

While working with the REPL, you safely can press Tab to make an indentation level, and although a Tab character can do it, using four spaces is the *strongly* recommended way. Many text editors know to put four spaces when you press Tab when editing a Python file. Whatever you do, never, I repeat, *never*, mix Tabs with spaces. In other programming languages, it may make the community dislike you, but in Python, it'll make your program fail with weird error messages.

Being practical, to reproduce what I did, simply type the class header, `def add_post(blog, new_post):`, press Enter, press Tab, type `return blog + [new_post]`, press Enter, press Enter again, and that's it. Let's see the function in action:

```
>>> blog = add_post(blog, "Fourth post")
>>> blog
['My first post', 'Python is cool', 'A new post.',
'Fourth post']
>>>
```

add_post takes two parameters. The first is the blog itself, and it gets assigned to blog. This is tricky. The blog inside the function is not the same as the blog outside the function. They are in different scopes. That's why the following:

```
>>> def add_post(blog, new_post):
...     blog = blog + [new_post]
```

doesn't work. blog is modified only inside the function. By now, you might know that new_post contains the post passed to the function.

Our blog is growing, and it is time to see that the posts are simply strings, but we want to have a title and a body. One way to do this is to use tuples, like this:

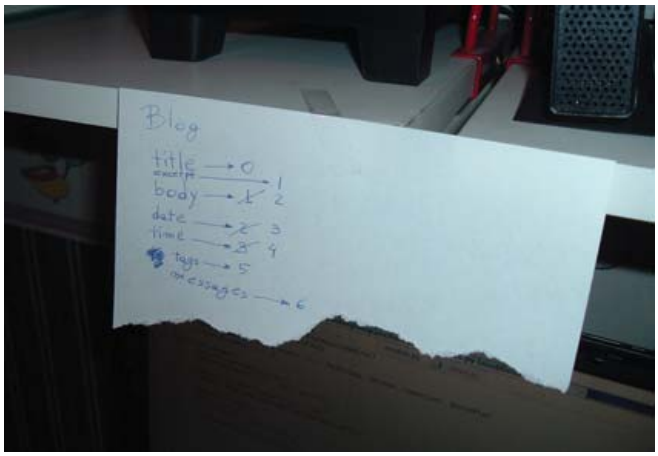


Figure 1. Index Handling the Hard Way

```
>>> blog = []
>>> blog = add_post(blog, ("New blog", "First post"))
>>> blog = add_post(blog, ("Cool", "Python is cool"))
>>> blog
[('New blog', 'First post'),
('Cool', 'Python and is cool')]
>>>
```

In the first line, I reset the blog to be an empty list. Then, I added two posts. See the double parentheses? The outside parentheses are part of the function call, and the inside parentheses are the creation of a tuple.

A tuple is created by parentheses, and its members are separated by commas. They are similar to lists, but semantically, they are different. For example, you can't update the members of a tuple. Tuples are used to build some kind of structure with a fixed set of elements. Let's see a tuple outside of our blog:

```
>>> (1,2,3)
(1, 2, 3)
```

Accessing each part of the posts is similar to accessing each part of the blog:

```
>>> blog[0][0]
'New blog'
>>> blog[0][1]
'This is my first post'
```

This might be a good solution if we want to store only a title and a body. But, how long until we want to add the date and time, excerpts, tags or messages? You may begin thinking you'll need to hang a sheet of paper on the wall, as shown in Figure 1, to remember the index of each field—not pleasant at all. To solve this problem, and some others, Python gives us object-oriented programming.

Object-Oriented Programming

Object-oriented programming was born more than 20 years ago so developers could separate each part of a computer program in a way similar to how objects are separated in the real world. Python models objects by using classes. A class is an abstract definition of what an object has and what an object can do. If this sounds foreign, don't worry, OOP (object-oriented programming) is difficult at first.

An example might help. A bridge is a structure that allows people or vehicles to cross an obstacle, such as a river, canal or railway. A bridge has some length, some width and even some color. It may allow vehicles or only persons. It may allow heavy vehicles or not. When I say "bridge", I am not defining any of those details. Bridge is a class. If I say Golden Gate, Le Pont de Normandie or Akashi-Kaikyo, I am naming particular bridges; they have some specific length, width, vehicle allowance and color. In OOP jargon, they are instances of bridge.

Back to our blog, let's create a class to model our post:

```
>>> class Post(object):
...     pass
...
>>>
```

We start with class, the keyword for creating new classes. Next comes the name of the class—in this case, Post. In parentheses, we have the super-classes—ignore that for now.

Here again, the prompt has changed from >>> to ..., and Python expects *something* in a class. Because we don't want to put anything in yet, we write `pass`, which is something, but in fact, it is nothing. Python knows when a class starts and ends because of the indentation, the same as with functions.

To reproduce what I did, simply type the class header, `class Post(object):`, press Enter, press Tab, type `pass`, press Enter, press Enter again, and that's it.

Now, we can create a Post:

```
>>> cool = Post()
>>> cool
<__main__.Post object at 0xb7ca642c>
```

Note that what is being printed when we evaluate a post is a generic representation for the object. We can set its title and body:

```
>>> cool.title = "Cool"
>>> cool.body = "Python is cool."
```

And retrieve them:

```
>>> cool.title
'Cool'
>>> cool.body
'Python is cool.'
```

Up to this point, a Post is like a simple container for anything you can imagine putting there. The problem with this is we may get lost as to what to put in it, or what not to put in it. Back to a sheet of paper? No! Although we can't stop making the posts a container in that way, we can put some methods there, so users have an idea of what a post may contain. To do this, we write our own methods in the class Post:

```
>>> class Post(object):
...     def set_title(self, title):
...         self._title = title
...     def get_title(self):
...         return self._title
...
>>>
```

Python modules are simple text files, and you can use any text editor you want.

Methods are like functions, but as they are in a class, they are associated with the objects of the class. This means different classes can have different methods with the same name. Just imagine the difference between bat.hit(ball) and stick.hit(drum).

Python has a convention that the first parameter (normally called self) to a method is the object on which we are calling the method. That means running `cool.set_title("Cool")` will set `self` to be `cool`, and `title` to be `"Cool"`. Running:

```
cool.set_title("Cool")
```

is the equivalent of:

```
cool._title = "Cool"
```

The leading underscore lets others know that we don't want them playing with it. It means "don't access _title; use get_title and set_title".

The previous interaction with the cool object can be rewritten as:

```
>>> cool = Post()
>>> cool.set_title("Cool")
>>> cool.set_body("Python is cool.")
>>> cool.get_title()
'Cool'
>>> cool.get_body()
'Python is cool.'
```

Writing the same set of methods for body should be easy now. But, be aware that you have to write the whole class in one go. Write the class header, the set_title and get_title methods, and then create your set_body and get_body methods. It may take you a couple of tries.

Files

As the Post class becomes bigger, you'll get tired of rewriting it every time you want to add a method. If you're tired already, that's a good sign. And besides, all that's in the REPL will be lost when we quit Python. We should start saving our work in files.

Python modules are simple text files, and you can use any text editor you want. As a programmer, you are going to spend most of your time with your editor, so take some time to choose one you really like and learn to use it well.

Emacs might not be the most beautiful editor, but for many programming tasks, it is awesome. (You could read that as "I don't like Emacs but it makes my life so much easier that I keep coming to it time after time".) Installing Emacs from source is beyond the scope of this article. As usual, with programs that are so popular, your distribution is likely to provide it. In Debian and its derivatives try:

```
apt-get install emacs
```

For Gentoo, the counterpart is:

```
emerge emacs
```

To achieve the magic I am going to show here, you need python-mode. In Debian:

```
apt-get install python-mode
```

In Gentoo:

```
emerge python-mode
```

Run Emacs. If you are serious about learning how to use it, now it is time to press Ctrl-H T, which in Emacs jargon means press Ctrl-H, release it and then press T. But, you can leave that for later, when you've finished reading this *Linux Journal* issue. For this article, I provide all the keystrokes you need.

Press Ctrl-X Ctrl-F (Ctrl-X, release, Ctrl-F) to visit a file. On the bottom of the Emacs window, you'll see the cursor waiting for you to type the path and filename. Type `blog.py` and press Enter. (Python modules should have the extension .py.) Now, you can start typing the Post class we programmed before. Emacs tries to be smart about indentation and places it where you are likely to want it. If you need a different indentation, simply press Tab and keep pressing it until you get the desired results.

On the top, you should have two menus: IM-Python and Python. The first one contains a list of classes and methods in the file you are editing. Click on Rescan if it doesn't show information you know is there. This is very useful when working with huge files. The second menu is even more useful, but explore and play with it later. For now, simply run `Start interpreter...` or press Ctrl-C !.

Suddenly the window is split, and you have an embedded Python interpreter below the file you are editing (Figure 2). And the fun is only beginning. Click on the file you are editing to set the focus on it. Run Import/reload file from the Python menu or press Ctrl-C Enter. Now, you're ready to test your code on the REPL, but be aware that

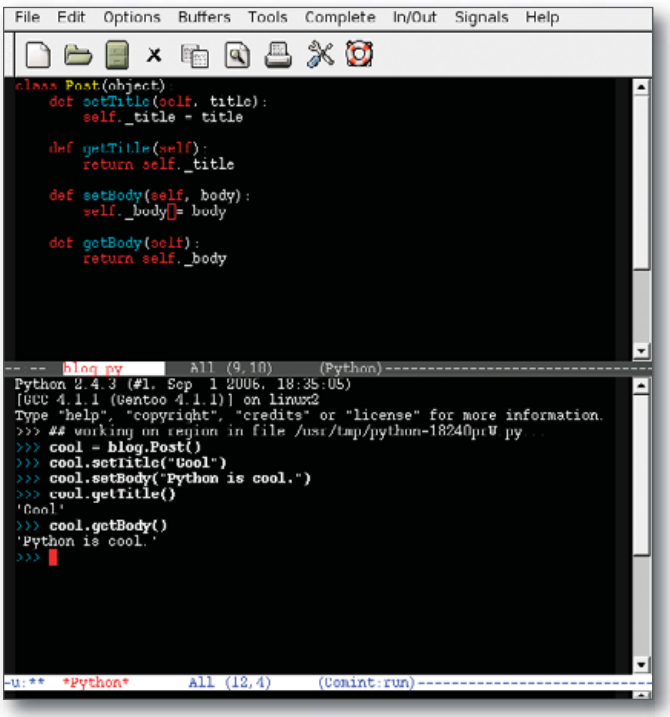


Figure 2. Testing the REPL

EmperorLinux

...where Linux & laptops converge

Portable

Since 1999, EmperorLinux has provided pre-installed Linux laptops to universities, corporations, government labs, and individual Linux enthusiasts. Our laptops range from full-featured ultra-portables to desktop replacements. All systems come with one year of Linux technical support by phone and e-mail, and full manufacturers' warranties apply.

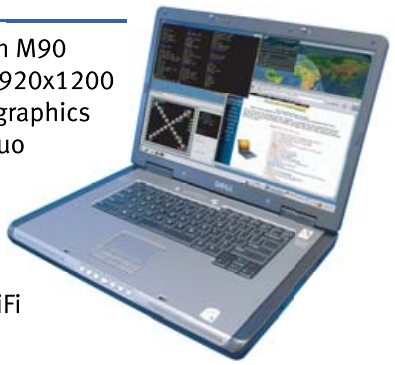
Toucan T60/T60ws

ThinkPad T60/T60ws by Lenovo
- Up to 15.4" WSXGA+ w/ X@1680x1050
- ATI Mobility FireGL V5200
- 1833–2333 MHz Core 2 Duo
- 512 MB–4 GB RAM
- 60–120 GB hard drive
- CDRW/DVD or DVD±RW
- 5.2–6.0 pounds
- 10/100/1000 Mbps ethernet
- 802.11a/b/g (54Mbps) WiFi
- **Starts at $1950**

Powerful

EmperorLinux specializes in the installation of Linux on a wide range of the finest laptops made by IBM, Lenovo, Dell, Sony, and Panasonic. We customize your choice of Linux distribution to your laptop and provide support for: ethernet, wireless, X-server, ACPI power management, USB, EVDO, PCMCIA, FireWire, CD/DVD/CDRW, sound, and more.

Rhino D820/M90



Dell Latitude D820/Precision M90
- Up to 17" WUXGA w/ X@1920x1200
- NVidia Quadro FX 3500M graphics
- 1667–2333 MHz Core 2 Duo
- 512 MB–4 GB RAM
- 60–160 GB hard drive
- CDRW/DVD or DVD±RW
- 6.3–8.6 pounds
- 802.11a/b/g (54Mbps) WiFi
- ExpressCard/EVDO
- **Starts at $1480**

Unique

EmperorLinux offers Linux laptops with unique features. Ruggedized Panasonic laptops are designed for harsh environments: drops, vibrations, sand, rain, and other extremes. ThinkPad tablet PCs are like other laptops, with an LCD digitizer for pen-based input both as a mouse and with pressure sensitivity for writing and drawing on-screen.

Raven X60 Tablet

ThinkPad X60 Tablet by Lenovo
- 12.1" SXGA+ w/ X@1400x1050
- 1667–1833 MHz Core Duo
- 1–4 GB RAM
- 80–120 GB hard drive
- 3.8 pounds
- Pen/stylus input to screen
- Dynamic screen rotation
- Handwriting recognition
- X60s laptops available
- **Starts at $2300**

www.EmperorLinux.com 1-888-651-6686

you'll have to add `blog.` before the name of the class, `Post`, because now the class is in the module blog. See Figure 2 for further reference.

You can, of course, do the same without Emacs. But for that, you need to learn how Python modules and packages are made. Set PYTHON_PATH, an environment variable, accordingly, and use the built-in function reload. With Emacs, you'll find iterating between coding and testing the code to be very fast. This speed can improve your performance and make programming more fun. In fact, Lisp programmers have been advocating this way of working for more than two decades.

Special Methods

Having to create an object and then set each of its members is not pleasant. It takes a lot of lines and is very error-prone—did I remember to set the tags? There's a better way to do it—using the initialization method.

This special method is called _ _init_ _, and the parameters you define it to take have to be passed in the creation of the object. A possible initialization method would be:

```
class Post(object):
    def __init__(self, title, body):
        self.set_title(title)
        self.set_body(body)
```

Simply add the _ _init_ _ definition to the file and reload it. We now can, and have to, set the title and body at initialization time:

```
>>> cool = blog.Post("Cool", "Python is cool")
>>> cool.get_title()
'Cool'
>>> cool.get_body()
'Python is cool'
>>>
```

Hint: to retrieve previous lines in the REPL inside Emacs use Alt-P.

There are other special methods. Remember how ugly it was to evaluate a Post itself? Let me remind you:

```
>>> cool
<blog.Post object at 0xb7c7e9ac>
```

We can solve that. There's another special method called _ _repr_ _, which is used to retrieve that string. Inside the Post class add:

```
def __repr__(self):
    return "Blog Post: %s" % self.get_title()
```

Reload the file, the same way you loaded it previously, and evaluate a post:

```
>>> ## working on region in file /usr/tmp/python...
>>> cool
<blog.Post object at 0xb7c7e9ac>
>>>
```

Oops! That's not what we wanted. The problem here is that the cool object was created with an older version of the Post class, so it doesn't have the new method. That is a very common mistake, and not being prepared for it can cause a lot of headaches. But, simply re-create the object, and you are set:

```
>>> ## working on region in file /usr/tmp/python...
>>> cool = blog.Post("Cool", "Python is cool")
>>> cool
Blog Post: Cool
>>>
```

That's better.

What Now?

Easy—wait for the next issue of *Linux Journal* for Part II of this tutorial. If you really want something to do now, start learning Emacs.■

José P. E. "Pupeno" Fernández has been programming since...at what age is a child capable of siting in a chair and reaching a keyboard? He has experimented with more languages than can be listed on this page. His Web site is at pupeno.com, and he always can be reached, unless you are a spammer, at pupeno@pupeno.com.

Resources

Python: **python.org**

Python Download: **python.org/download**

Python 2.4.3: **www.python.org/ftp/python/2.4.3/Python-2.4.3.tgz**

Hear Yourself Think Again!

WhisperStation™

Cool... Fast... Silent!

For 64-bit HPC, Gaming and Graphic Design Applications

Originally designed for a group of power hungry, demanding engineers in the automotive industry, WhisperStation™ incorporates two dual core AMD Opteron™ or Intel® EM64T™ processors, ultra-quiet fans and power supplies, plus internal sound-proofing that produce a powerful, but silent, computational platform. The WhisperStation™ comes standard with 2 GB high speed memory, an NVIDIA e-GeForce or Quadro PCI Express graphics adapter, and 20" LCD display. It can be configured to your exact hardware specification with any Linux distribution. RAID is also available. WhisperStation™ will also make a system administrator very happy, when used as a master node for a Microway cluster! Visit www.microway.com for more technical information.

Experience the "Sound of Silence".
Call our technical sales team at 508-746-7341 and design your personalized WhisperStation™ today.

AMD 64 Opteron

intel inside XEON

Microway®
Technology you can count on℠

Asynchronous Database Access with Qt 4.x

How to code around the default synchronous database access in Qt 4.

Dave Berton

The database support in Qt 4.x is quite robust. The library includes drivers for Oracle, PostgreSQL, SQLite and many other relational databases. Out of the box, the Qt database library also contains bindings for many widgets and provides data types for the transparent handling of result sets coming from a database. But, your application can pay a price for these conveniences. All database access is synchronous by default, which means that intensive and time-consuming SQL queries normally will lock up the UI unless precautions are taken. Using stored procedures on the server can sometimes help the situation; however, this is not always possible or desirable. And often, the length and cost of the queries generated by your application simply cannot be known in advance, so the door is left open for undesirable UI behavior. People don't want their application to "lock up" at odd moments; however, this is the default behavior, and so we must contend with it.

Fortunately, Qt 4.x also has robust support for multithreaded programming. By placing the heavy-duty database work in separate threads, the UI is free to respond to the user normally, without ungraceful interruptions. As with all concurrent programming, however, you must take precautions to ensure the correct sequence of interactions between threads. For example, when sharing data among threads, guard it properly using mutexes. When communicating between threads, consider carefully how the interaction will behave, and in what sequence. In addition, when utilizing a database connection within a thread separate from the UI thread, you must pay attention to some extra caveats. A proper implementation that keeps certain things in mind will make significant improvements in the UI behavior and responsiveness of a database application.

Thread Strategies

There are several ways to distribute the database load to separate threads of execution. Fortunately, all of them share the same characteristics when it comes to the details of creating and using a database connection properly. The primary consideration is to use a database connection only within the thread that created it. For regular synchronous applications, the default behavior is fine. The QSqlDatabase::addDatabase() static function creates a database connection within the context of the application's main UI thread. Queries executed within this same thread will then cause blocking behavior. This is to be expected.

In order to run queries in parallel with the main UI thread, so that they do not interrupt the main event processing loop, a database connection must be established in the thread in which the query executes, which should be separate from the main UI thread. However you structure the threading in your application, your design must be able to establish a connection within the context of each thread that will be performing database work.

For example, creating a thread pool in which a few threads handle the load of querying the database in a round-robin fashion (without the overhead of creating and destroying threads all the time) will push the time-consuming work outside the main event loop. Or, depending on the needs of your application, you simply can spawn threads on an as-needed basis to perform database work. In either case, you must create a connection per thread.

Listing 1. Create two instances of QThread, one for queries, another for updates.

```
class QueryThread : public QThread
{
  public:
    QueryThread( QObject* parent = 0 )
    {
      //...
    }
    void run()
    {
        QSqlDatabase db = QSqlDatabase::addDatabase(
        ➥"QPSQL", "querythread" );
        // use 'db' here
    }
};

class UpdateThread : public QThread
{
  public:
    UpdateThread( QObject* parent = 0 )
    {
      //...
    }
    void run()
    {
        QSqlDatabase db = QSqlDatabase::addDatabase(
        ➥"QPSQL", "updatethread" );
        // use 'db' here
    }
};
```

There is a further limitation (imposed by most of the underlying database-specific libraries used by Qt). As a general rule, connections cannot be shared by multiple threads. This means you cannot simply create a pool of connections on startup and hand them out to various threads as needed. Instead, each thread must establish and maintain its own connection, within its own context. To do otherwise is undefined, and probably disastrous. Multiple separate connections can be established in each thread by using the name parameter of the QSqlDatabase::addDatabase() static function, as shown in Listing 1.

In Listing 1, the thread objects establish two different database connections. Each connection is named separately, so that QSqlDatabase can maintain them properly in its internal list. And, most important, each connection is established within the separate thread of execution of each object—the run() method is invoked by QThread::start() once the new thread of execution is launched. The mechanism provided by QSqlDatabase to create new connections is thread-safe. Listing 2 shows another example of a more generic approach.

The pseudo-code in Listing 2 creates and starts two worker threads. Each thread establishes a named connection to the database and waits for work to do. Each thread always will deal only with its

Listing 2. A More Generic Approach with a Single Instance of QThread Used Twice

```
class QueryThread : public QThread
{
  public:
    QueryThread( const QString& name )
      : m_connectionname(name)
    {
      //...
    }
    void run()
    {
      QSqlDatabase db =QSqlDatabase::addDatabase( "QPSQL", m_connectionname );
      //...
      db.open();
      forever
      {
        // wait for work
        // and then execute it...
      }
    }
}

void main()
{
  QApplication app(...);
  MainWin mw;
  QueryThread db1("queries");
  db1.start();
  QueryThread db2("updates")
  db2.start();
  //...
  mw.show();
  app.exec();
}
```

Communicating between threads now becomes a matter of connecting signals from one thread to the slots in another, and the mutexing and thread-safety issues of exchanging data between threads are handled by Qt.

own database connection, which is never shared or visible outside the worker thread object.

Setting up thread-specific connections and running queries within that thread is only the first part of the problem. A decision needs to be made regarding how data is shuffled back and forth between the worker threads and the main application UI thread. Additional methods will give the thread object some database work to perform, and those methods will themselves need to be thread-safe.

Moving Data between Threads

You should observe all of the usual caveats surrounding the sharing of data between different threads of execution, including the proper uses of mutexes and wait conditions. In addition, there is an added complication regarding the size of the data, which potentially can be extremely large in the case of result sets returned from the database.

Qt provides several specialized mechanisms for sending data between threads. You can post events manually to any object in any thread using the thread-safe function QCoreApplication::postEvent(). The events will be dispatched automatically by the event loop of the thread where the destination object was created. To create an event loop within each thread, use QThread::exec(). Using this method, threads are given "work" to do in the form of events.

Listing 3. Code That Shares Information from Worker Thread to Application Thread via Event

```
class WorkEvent : public QEvent
{
  public:
    enum { Type = User + 1 }
    WorkEvent( const QString& query )
      : QEvent(Type)
      , m_query(query)
    {}

    QString query() const
    {
      return m_query;
    }
  private:
    QString m_query;
};

QueryThread thread;
thread.start();
//...
WorkEvent* e = new WorkEvent("select salary from employee
 ➥where name='magdalena';");
  app.postEvent( &thread, e );
  //...
```

QCoreApplication passes these events in a thread-safe manner from the application thread to the worker thread, and they will be handled by the worker thread within its own execution context. This is critical, because the worker thread will be utilizing its database connection only from within its own context (Listing 3).

This method works in the other direction as well. Events posted by individual worker threads will show up back in the main UI event loop for handling, within the context of the UI's thread of execution. These events can, for example, contain the result of a query or database update. This approach is convenient, as the worker threads can simply "post it and forget it", and Qt will take care of the inter-thread communication, mutexing and memory management for you.

Constructing all the necessary events and event handlers for this type of system has advantages—most notably compile-time type

Listing 4. Sharing information across threads is cleaner with signals and slots.

```
class Worker; // forward decl
class QueryThread : public QThread
{
  QueryThread();
  signals:
    void queryFinished( const QList<QSqlRecord>& records );
  slots:
    void slotExecQuery( const QString& query );
  signals:
    void queue( const QString& query );
  private:
    Worker* m_worker;
};

int main()
{
  QApplication app;

  MainWin mw;

  QueryThread t;
  t.start();

  connect(&mw, SIGNAL( execQuery(const QString&)),
          &t, SLOT( slotExecQuery(const QString&)));

  connect(&t, SIGNAL( queryFinished(const QList<QSqlRecord>&)),
          &mw, SLOT( slotDisplayResults(const QList<QSqlRecord>&)));

  mw.show();
  return app.exec();
}
```

checking. However, getting all the events designed and dealt with properly can be an onerous task, especially if your application has multiple types of database queries to perform, with multiple return types, each of which needing an associated event and event handler and so forth.

Fortunately, Qt permits signals and slots to be connected across threads—as long as the threads are running their own event loops. This is a much cleaner method of communication compared to sending and receiving events, because it avoids all the bookkeeping and intermediate QEvent-derived classes that become necessary in any nontrivial application. Communicating between threads now becomes a matter of connecting signals from one thread to the slots in another, and the mutexing and thread-safety issues of exchanging data between threads are handled by Qt.

Why is it necessary to run an event loop within each thread to which you want to connect signals? The reason has to do with the inter-thread communication mechanism used by Qt when connecting signals from one thread to the slot of another thread. When such a connection is made, it is referred to as a queued connection. When signals are emitted through a queued connection, the slot is invoked the next time the destination object's event loop is executed. If the slot had instead been invoked directly by a signal from another thread, that slot would execute in the same context as the calling thread. Normally, this is not what you want (and especially not what you want if you are using a database connection, as the database connection can be used only by the thread that created it). The queued connection properly dispatches the signal to the thread object and invokes its slot in its own context by piggy-backing on the event system. This is precisely what we want for inter-thread communication in which some of the threads are handling database connections. The Qt signal/slot mechanism is at root an implementation of the inter-thread event-passing scheme outlined above, but with a much cleaner and easier-to-use interface.

For example, you can create two simple connections between the main UI object and a worker thread object; one to add a query to the worker thread and another to report back the results. This simple setup, only a few lines of code, establishes the main communication mechanism for an asynchronous database application (Listing 4).

Here, the MainWin and the QueryThread objects communicate directly with one another via signals and slots in the usual way. The trick here is that the QueryThread object, behind the scenes, utilizes a Worker object to perform all the work. Why is this extra level of indirection necessary? Because we want to dispatch the work to an object that is associated with a completely separate thread of execution.

Listing 5. An extra level of indirection makes execution more asynchronous.

```
class Worker : public QObject
{
  Q_OBJECT

   public:
    Worker( QObject* parent = 0);
    ~Worker();

  public slots:
    void slotExecute( const QString& query );

  signals:
    void results( const QList<QSqlRecord>& records );

   private:
     QSqlDatabase m_database;
};

void QueryThread::run()
{
  // Create worker object within the context of the new thread
  m_worker = new Worker();

  // forward to the worker: a 'queued connection'!
  connect( this, SIGNAL( queue( const QString& ) ),
           m_worker, SLOT( slotExecute( const QString& ) ) );
  // forward a signal back out
  connect( m_worker, SIGNAL( results( const QList<QSqlRecord>& ) ),
           this, SIGNAL( queryFinished( const QList<QSqlRecord>& ) ) );

  exec();  // start our own event loop
}

void QueryThread::execute( const QString& query )
{
  emit queue( query ); // queues to worker
}
```

Notice above that QueryThread is instantiated and start()ed within the main thread of execution; therefore, QueryThread will "belong" to the application's main thread. Connecting signals from the application's widgets to its slots will be via "direct" connections—they will be invoked immediately by Qt, in the usual way, and thus be blocking, synchronous function calls. Instead, we are interested in "pushing" the signals into slots that are running in a separate thread of execution entirely, and this is where the internal Worker class comes into play (Listing 5).

Utilizing this Worker class internally, QueryThread can dispatch SQL queries properly to a separate thread by employing the convenient inter-thread signal/slot queued connections provided by Qt. QueryThread encapsulates the idea of a thread, by being derived from QThread and being able to start() a new thread of execution, and it also encapsulates the idea of a worker (by exposing convenient methods that perform database work, which are internally dispatched to a separate thread of execution). So, instead of tangling with all the necessary events and event handlers to accomplish the same task, the Qt metaobject system rigs up all the necessary code for you and exposes the connect() function to trigger it all. Signals can be emitted at any time and will be dispatched to the destination object correctly, within that object's context.

It is important to note that, above, the execute() method of QueryThread is intended to be invoked synchronously by the main application. QueryThread presents this method not only as a convenience; indeed, it also encapsulates and hides all the queued connection details from the user of the QueryThread class. When execute() is invoked, QueryThread simply emits it as a signal to the worker. Because the worker is an object "living" in a separate thread, Qt will "queue" the connection and pass it through the event loop so that it arrives in the proper execution context of the worker—essential so the worker can utilize its database connection according to the rules that are laid out in the beginning of this article. Finally, any signals coming from the worker are "forwarded" back out through the QueryThread interface—another convenience for the users of QueryThread, which also serves to hide all the details of the Worker class from those who really don't need to know about it in the first place.

If the sizes of your queries are gigantic, or potentially can be gigantic, you should consider a different strategy for dealing with the result sets. For example, writing the results out to disk before passing them back to the UI thread will reduce the memory load of your application, at the cost of some runtime speed. However, if you are already dealing with massive queries, the speed hit likely will be minor in comparison with the overall execution time of the query. Because all database queries execute in a completely separate thread from the UI, users will not perceive any significant lag. Store the intermediate results in a text file, or for maximum flexibility, in a local SQLite database.

One element of this puzzle still needs to be solved. The queued connection mechanism relies on the thread's event loop to invoke a slot within that thread's context properly, as discussed previously. However, in order to marshal the data necessary to dispatch the event to another thread of execution, the Qt object system must be made aware of any custom data types used in the signal/slot declaration. Failure to register custom data types will cause the queued connection to fail, because in order to dispatch the event, a copy of each of the signal's parameters must be made. Fortunately, it is a simple matter to "register" additional types, as long as those types have public constructors, copy constructors and destructors (Listing 6).

Listing 6. You must register any custom data types in order to share them.

```
// first, make the object system aware
qRegisterMetaType< QList<QSqlRecord> >("QList<QSqlRecord>");
// now set up the queued connection
connect( m_worker, SIGNAL( results( const QList<QSqlRecord>& ) ),
     this, SIGNAL( queryFinished( const QList<QSqlRecord>& ) ) );
```

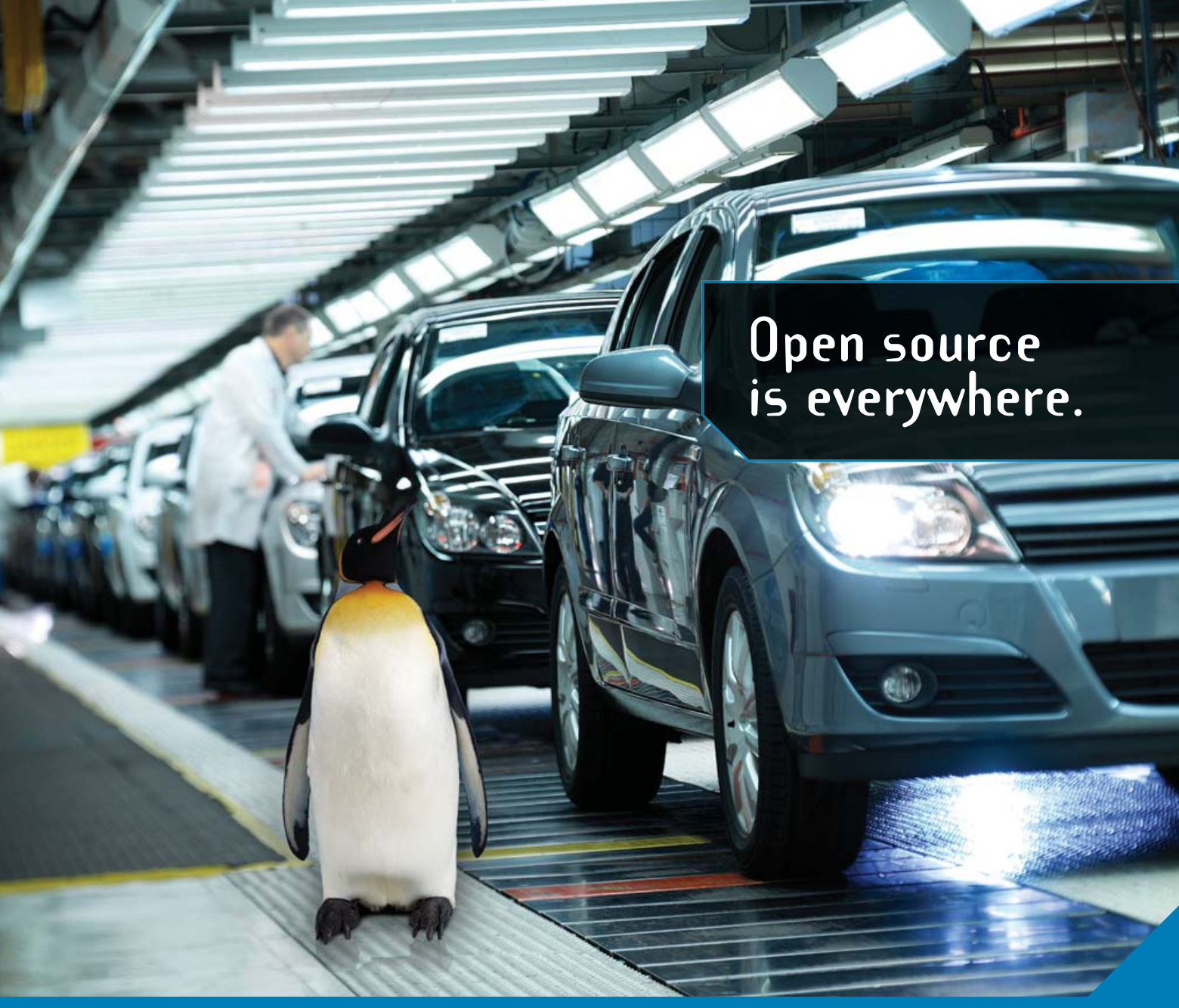
The sample application provided with this article implements the strategy outlined above, in which queries are executing in parallel with the rest of the application [the application is available for download from the *Linux Journal* FTP site, **ftp.linuxjournal.com/pub/lj/issue158/9602.tgz**]. The UI is not disturbed while the queries are underway. A queued connection is created between the QueryThread interface and the encapsulated worker thread after the appropriate types are registered with the Qt metaobject system. This allows the separate threads to communicate safely with one another, with minimal overhead and code complexity. The sample application was tested with SQLite and PostgreSQL; however, it will work with any database connection supported by Qt that enforces the same connection-per-thread limitation.

Summary

The following points should be kept in mind when designing asynchronous database applications with Qt:

- Create a database connection per thread. Use the name parameter of the thread-safe QSqlDatabase::addDatabase() method in order to distinguish various database connections.
- Encapsulate the database connection within worker thread objects as much as possible. Never share a database connection with another thread. Never use a database connection from any thread other than the one that created it.
- Manage communication between threads using the tools provided by Qt. In addition to QMutex, QSemaphore and QWaitCondition, Qt provides much more direct mechanisms: events and signals/slots. The implementation of signals/slots across thread boundaries relies on events; therefore, ensure that your threads start their own event loop using QThread::exec().
- Register unknown types with the Qt metaobject system. Any unknown types cannot be marshaled properly without first invoking qRegisterMetaType(). This enables a queued connection to invoke a slot in a separate thread within that thread's context using new types.
- Utilize queued connections to communicate between the application and the database threads. The queued connection provides all the advantages for dealing with asynchronous database connections, but maintains a simple and familiar interface using QObject::connect().■

Dave Berton is a professional programmer working for Eventide, Inc. He welcomes your comments at dberton@eventide.com.



Open source is building momentum. Find out why at LinuxWorld.

From virtualization to mobile applications, you'll learn what's fueling the movement to open source. More companies are switching to open source than ever. For example, a leading Japanese auto manufacturer uses it to connect 10,000 users at 1,300 car dealers to its U.S. headquarters. By utilizing open source on high performance servers, they saved more than $650,000 upon implementation. The system now handles mission-critical functions including parts ordering, warranties, sales transactions and repairs. To accelerate your plans, register at www.linuxworldexpo.com.

LinuxWorld® CONFERENCE & EXPO

August 6-9, 2007
Moscone Center - San Francisco

Register now at
www.linuxworldexpo.com

Platinum Sponsors

ACCESS ORACLE®

Gold Sponsor

Novell®

Silver Sponsor

APPRO HPC Cluster Solutions

Validate an E-Mail Address with PHP, the Right Way

Develop a working PHP function to validate e-mail addresses.

DOUGLAS LOVELL

The Internet Engineering Task Force (IETF) document, RFC 3696, "Application Techniques for Checking and Transformation of Names" by John Klensin, gives several valid e-mail addresses that are rejected by many PHP validation routines. The addresses: Abc\@def@example.com, customer/department=shipping@example.com and !def!xyz%abc@example.com are all valid. One of the more popular regular expressions found in the literature rejects all of them:

```
"^[_a-z0-9-]+(\.[_a-z0-9-]+)*@[a-z0-9-]+(\.[a-z0-9-]+)
➥*(\.[a-z]{2,3})$"
```

This regular expression allows only the underscore (_) and hyphen (-) characters, numbers and lowercase alphabetic characters. Even assuming a preprocessing step that converts uppercase alphabetic characters to lowercase, the expression rejects addresses with valid characters, such as the slash (/), equal sign (=), exclamation point (!) and percent (%). The expression also requires that the highest-level domain component has only two or three characters, thus rejecting valid domains, such as .museum.

Another favorite regular expression solution is the following:

```
"^[a-zA-Z0-9_.-]+@[a-zA-Z0-9-]+.[a-zA-Z0-9-.]+$"
```

This regular expression rejects all the valid examples in the preceding paragraph. It does have the grace to allow uppercase alphabetic characters, and it doesn't make the error of assuming a high-level domain name has only two or three characters. It allows invalid domain names, such as example..com.

Listing 1 shows an example from PHP Dev Shed (**www.devshed.com/c/a/PHP/Email-Address-Verification-with-PHP/2**). The code contains (at least) three errors. First, it fails to recognize many valid e-mail address characters, such as percent (%). Second, it splits the e-mail address into user name and domain parts at the at sign (@). E-mail addresses that contain a quoted at sign, such as Abc\@def@example.com will break this code. Third, it fails to check for host address DNS records. Hosts with a type A DNS entry will accept e-mail and may not necessarily publish a type MX entry. I'm not picking on the author at PHP Dev Shed. More than 100 reviewers gave this a four-out-of-five-star rating.

Listing 1. An Incorrect E-mail Validation

```
function checkEmail($email) {
  if(preg_match("/^([a-zA-Z0-9])+([a-zA-Z0-9\._-])
  ➥*@([a-zA-Z0-9_-])+([a-zA-Z0-9\._-]+)+$/",
                $email)){
    list($username,$domain)=split('@',$email);
    if(!checkdnsrr($domain,'MX')) {
      return false;
    }
    return true;
  }
  return false;
```

One of the better solutions comes from Dave Child's blog at ILoveJackDaniel's (**ilovejackdaniels.com**), shown in Listing 2 (**www.ilovejackdaniels.com/php/email-address-validation**). Not only does Dave love good-old American whiskey, he also did some homework, read RFC 2822 and recognized the true range of characters valid in an e-mail user name. About 50 people have commented on this solution at the site, including a few corrections that have been incorporated into the original solution. The only major flaw in the code collectively developed at ILoveJackDaniel's is that it fails to allow for quoted characters, such as \@, in the user name. It will reject an address with more than one at sign, so that it does not get tripped up splitting the user name and domain parts using `explode("@", $email)`. A subjective criticism is that the code expends a lot of effort checking the length of each component of the domain portion—effort better spent simply trying a domain lookup. Others might appreciate the due diligence paid to checking the domain before executing a DNS lookup on the network.

Requirements

IETF documents, RFC 1035 "Domain Implementation and Specification", RFC 2234 "ABNF for Syntax Specifications", RFC 2821 "Simple Mail Transfer Protocol", RFC 2822 "Internet Message Format", in addition to RFC 3696 (referenced earlier), all contain information relevant to e-mail address validation. RFC 2822 supersedes RFC 822 "Standard for ARPA Internet Text Messages" and makes it obsolete.

Following are the requirements for an e-mail address, with relevant references:

1. An e-mail address consists of local part and domain separated by an at sign (@) character (RFC 2822 3.4.1).

2. The local part may consist of alphabetic and numeric characters, and the following characters: !, #, $, %, &, ', *, +, -, /, =, ?, ^, _, `, {, |, } and ~, possibly with dot separators (.), inside, but not at the start, end or next to another dot separator (RFC 2822 3.2.4).

3. The local part may consist of a quoted string—that is, anything within quotes ("), including spaces (RFC 2822 3.2.5).

4. Quoted pairs (such as \@) are valid components of a local part, though an obsolete form from RFC 822 (RFC 2822 4.4).

Listing 2. A Better Example from ILoveJackDaniel's

```
function check_email_address($email) {
  // First, we check that there's one @ symbol,
  // and that the lengths are right.
  if (!ereg("^[^@]{1,64}@[^@]{1,255}$", $email)) {
    // Email invalid because wrong number of characters
    // in one section or wrong number of @ symbols.
    return false;
  }
  // Split it into sections to make life easier
  $email_array = explode("@", $email);
  $local_array = explode(".", $email_array[0]);
  for ($i = 0; $i < sizeof($local_array); $i++) {
    if
(!ereg("^(([A-Za-z0-9!#$%&'*+/=?^_`{|}~-][A-Za-z0-9!#$%&
➥'*+/=?^_`{|}~\.-]{0,63})|(\"[^(\\|\")]{0,62}\"))$",
$local_array[$i])) {
      return false;
    }
  }
  // Check if domain is IP. If not,
  // it should be valid domain name
  if (!ereg("^\[?[0-9\.]+\]?$", $email_array[1])) {
    $domain_array = explode(".", $email_array[1]);
    if (sizeof($domain_array) < 2) {
        return false; // Not enough parts to domain
    }
    for ($i = 0; $i < sizeof($domain_array); $i++) {
      if
(!ereg("^(([A-Za-z0-9][A-Za-z0-9-]{0,61}[A-Za-z0-9])|
➥([A-Za-z0-9]+))$",
$domain_array[$i])) {
        return false;
      }
    }
  }
  return true;
}
```

5. The maximum length of a local part is 64 characters (RFC 2821 4.5.3.1).

6. A domain consists of labels separated by dot separators (RFC1035 2.3.1).

7. Domain labels start with an alphabetic character followed by zero or more alphabetic characters, numeric characters or the hyphen (-), ending with an alphabetic or numeric character (RFC 1035 2.3.1).

8. The maximum length of a label is 63 characters (RFC 1035 2.3.1).

9. The maximum length of a domain is 255 characters (RFC 2821 4.5.3.1).

10. The domain must be fully qualified and resolvable to a type A or type MX DNS address record (RFC 2821 3.6).

Requirement number four covers a now obsolete form that is arguably permissive. Agents issuing new addresses could legitimately disallow it; however, an existing address that uses this form remains a valid address.

The standard assumes a seven-bit character encoding, not multi-byte characters. Consequently, according to RFC 2234, "alphabetic" corresponds to the Latin alphabet character ranges a–z and A–Z. Likewise, "numeric" refers to the digits 0–9. The lovely international standard Unicode alphabets are not accommodated—not even encoded as UTF-8. ASCII still rules here.

Developing a Better E-mail Validator

That's a lot of requirements! Most of them refer to the local part and domain. It makes sense, then, to start with splitting the e-mail address around the at sign separator. Requirements 2–5 apply to the local part, and 6–10 apply to the domain.

The at sign can be escaped in the local name. Examples are, Abc\@def@example.com and "Abc@def"@example.com. This means an explode on the at sign, `$split = explode("@", $email);` or another similar trick to separate the local and domain parts will not always work. We can try removing escaped at signs, `$cleanat = str_replace("\\@", "");`, but that will miss pathological cases, such as Abc\\@example.com. Fortunately, such escaped at signs are not allowed in the domain part. The last occurrence of the at sign must definitely be the separator. The way to separate the local and domain parts, then, is to use the strrpos function to find the last at sign in the e-mail string.

Listing 3 provides a better method for splitting the local part and domain of an e-mail address. The return type of strrpos will be boolean-valued false if the at sign does not occur in the e-mail string.

Let's start with the easy stuff. Checking the lengths of the local part and domain is simple. If those tests fail, there's no need to do the more complicated tests. Listing 4 shows the code for making the length tests.

Listing 3. Splitting the Local Part and Domain

```
$isValid = true;
$atIndex = strrpos($email, "@");
if (is_bool($atIndex) && !$atIndex)
{
   $isValid = false;
}
else
{
   $domain = substr($email, $atIndex+1);
   $local = substr($email, 0, $atIndex);
   // ... work with domain and local parts
}
```

Listing 4. Length Tests for Local Part and Domain

```
$localLen = strlen($local);
$domainLen = strlen($domain);
if ($localLen < 1 || $localLen > 64)
{
   // local part length exceeded
   $isValid = false;
}
else if ($domainLen < 1 || $domainLen > 255)
{
   // domain part length exceeded
   $isValid = false;
}
```

Listing 5. Partial Test for Valid Local Part Content

```
if (!preg_match('/^(\\\\.|[A-Za-z0-9!#%&`_=\\/$\'*+?^{}|~.-])+$/',
                str_replace("\\\\","",$local)))
{
   // character not valid in local part unless
   // local part is quoted
   if (!preg_match('/^"(\\\\"|[^"])+"$/',
                   str_replace("\\\\","",$local)))
   {
      $isValid = false;
   }
}
```

Listing 6. Check for dot placement in the local part.

```
if ($local[0] == '.' || $local[$localLen-1] == '.')
{
   // local part starts or ends with '.'
   $isValid = false;
}
else if (preg_match('/\\.\\./', $local))
{
   // local part has two consecutive dots
   $isValid = false;
}
```

Listing 7. Domain Checks

```
if (!preg_match('/^[A-Za-z0-9\\-\\.]+$/', $domain))
{
   // character not valid in domain part
   $isValid = false;
}
else if (preg_match('/\\.\\./', $domain))
{
   // domain part has two consecutive dots
   $isValid = false;
}
else if (!(checkdnsrr($domain,"MX") || checkdnsrr($domain, "A")))
{
   // domain not found in DNS
   $isValid = false;
}
```

Listing 8. Test the e-mail validation function.

```
<?php
require("validEmail.php"); // your favorite here

function testEmail($email)
{
  echo $email;
  $pass = validEmail($email);
  if ($pass)
  {
    echo " is valid.\n";
  }
  else
  {
    echo " is not valid.\n";
  }
  return $pass;
}

$pass = true;
echo "All of these should succeed:\n";
$pass &= testEmail("dclo@us.ibm.com");
$pass &= testEmail("abc\\@def@example.com");
$pass &= testEmail("abc\\\\@example.com");
$pass &= testEmail("Fred\\ Bloggs@example.com");
$pass &= testEmail("Joe.\\\\Blow@example.com");
$pass &= testEmail("\"Abc@def\"@example.com");
$pass &= testEmail("\"Fred Bloggs\"@example.com");
$pass &= testEmail("customer/department=shipping@example.com");
$pass &= testEmail("\$A12345@example.com");
$pass &= testEmail("!def!xyz%abc@example.com");
$pass &= testEmail("_somename@example.com");
$pass &= testEmail("user+mailbox@example.com");
$pass &= testEmail("peter.piper@example.com");
$pass &= testEmail("Doug\\ \\\"Ace\\\"\\ Lovell@example.com");
$pass &= testEmail("\"Doug \\\"Ace\\\" L.\"@example.com");
echo "\nAll of these should fail:\n";
$pass &= !testEmail("abc@def@example.com");
$pass &= !testEmail("abc\\\\@def@example.com");
$pass &= !testEmail("abc\\@example.com");
$pass &= !testEmail("@example.com");
$pass &= !testEmail("doug@");
$pass &= !testEmail("\"qu@example.com");
$pass &= !testEmail("ote\"@example.com");
$pass &= !testEmail(".dot@example.com");
$pass &= !testEmail("dot.@example.com");
$pass &= !testEmail("two..dot@example.com");
$pass &= !testEmail("\"Doug \"Ace\" L.\"@example.com");
$pass &= !testEmail("Doug\\ \\\"Ace\\\"\\ L\\.@example.com");
$pass &= !testEmail("hello world@example.com");
$pass &= !testEmail("gatsby@f.sc.ot.t.f.i.tzg.era.l.d.");
echo "\nThe email validation ";
if ($pass)
{
   echo "passes all tests.\n";
}
else
{
   echo "is deficient.\n";
}
?>
```

Now, the local part has one of two forms. It may have a begin and end quote with no unescaped embedded quotes. The local part, Doug \"Ace\" L. is an example. The second form for the local part is, (a+(\.a+)*), where a stands for a whole slew of allowable characters. The second form is more common than the first; so, check for that first. Look for the quoted form after failing the unquoted form.

Characters quoted using the back slash (\@) pose a problem. This form allows doubling the back-slash character to get a back-slash character in the interpreted result (\\). This means we need to check for an odd number of back-slash characters quoting a non-back-slash character. We need to allow \\\\\@ and reject \\\\@.

It is possible to write a regular expression that finds an odd number of back slashes before a non-back-slash character. It is possible, but not pretty. The appeal is further reduced by the fact that the back-slash character is an escape character in PHP strings and an escape character in regular expressions. We need to write four back-slash characters in the PHP string representing the regular expression to show the regular expression interpreter a single back slash.

A more appealing solution is simply to strip all pairs of back-slash characters from the test string before checking it with the regular expression. The str_replace function fits the bill. Listing 5 shows a test for the content of the local part.

The regular expression in the outer test looks for a sequence of allowable or escaped characters. Failing that, the inner test looks for a sequence of escaped quote characters or any other character within a pair of quotes.

If you are validating an e-mail address entered as POST data, which is likely, you have to be careful about input that contains back-slash (\), single-quote (') or double-quote characters ("). PHP may or may not escape those characters with an extra back-slash character wherever they occur in POST data. The name for this behavior is magic_quotes_gpc, where gpc stands for get, post, cookie. You can have your code call the function, get_magic_quotes_gpc(), and strip the added slashes on an affirmative response. You also can ensure that the PHP.ini file disables this "feature". Two other settings to watch for are magic_quotes_runtime and magic_quotes_sybase.

The two regular expressions in Listing 5 are appealing because they are relatively easy to comprehend and don't require repetition of the allowable character group, [A-Za-z0-9!#%&`_=\/$\'*+?^{}|~.-]. Here's a test for you. Why does the character group require two back-slash characters before the forward slash and one back-slash character before the single quote?

One deficiency of the outer test of Listing 5 is that it passes local part strings that include dots anywhere in the string. Requirement number two states that dots can't start or end the local part, and they can't appear together two or more times. We could address this by expanding the outer regular expression into form ^(a+(\.a+)+)$, where a is (\\\\.|[A-Za-z0-9!#%&`_=\/$\'*+?^{}|~-]). We could, but that leads to a long, hard-to-read, repetitive expression that's difficult to believe in. It's clearer to add the simple checks shown in Listing 6.

Listing 9. A Complete E-mail Validation Function

```
/**
Validate an email address.
Provide email address (raw input)
Returns true if the email address has the email
address format and the domain exists.
*/
function validEmail($email)
{
   $isValid = true;
   $atIndex = strrpos($email, "@");
   if (is_bool($atIndex) && !$atIndex)
   {
      $isValid = false;
   }
   else
   {
      $domain = substr($email, $atIndex+1);
      $local = substr($email, 0, $atIndex);
      $localLen = strlen($local);
      $domainLen = strlen($domain);
      if ($localLen < 1 || $localLen > 64)
      {
         // local part length exceeded
         $isValid = false;
      }
      else if ($domainLen < 1 || $domainLen > 255)
      {
         // domain part length exceeded
         $isValid = false;
      }
      else if ($local[0] == '.' || $local[$localLen-1] == '.')
      {
         // local part starts or ends with '.'
         $isValid = false;
      }
      else if (preg_match('/\\.\\./', $local))
      {
         // local part has two consecutive dots
         $isValid = false;
      }
      else if (!preg_match('/^[A-Za-z0-9\\-\\.]+$/', $domain))
      {
         // character not valid in domain part
         $isValid = false;
      }
      else if (preg_match('/\\.\\./', $domain))
      {
         // domain part has two consecutive dots
         $isValid = false;
      }
      else if
(!preg_match('/^(\\\\.|[A-Za-z0-9!#%&`_=\\/$\'*+?^{}|~.-])+$/',
                  str_replace("\\\\","",$local)))
      {
         // character not valid in local part unless
         // local part is quoted
         if (!preg_match('/^"(\\\\"|[^"])+"$/',
             str_replace("\\\\","",$local)))
         {
            $isValid = false;
         }
      }
      if ($isValid && !(checkdnsrr($domain,"MX") ||
➥checkdnsrr($domain,"A")))
      {
         // domain not found in DNS
         $isValid = false;
      }
   }
   return $isValid;
}
```

The local part is a wrap. The code now checks all local part requirements. Checking the domain will complete the e-mail validation. The code could check all of the labels in the domain separately, as does the whiskey-loving code shown in Listing 2, but, as hinted earlier, the solution presented here allows the DNS check to do most of the domain validation work.

Listing 7 makes a cursory check to ensure only valid characters in the domain part, with no repeated dots. It goes on to make DNS lookups for MX and A records. It makes the check for the A record only if the MX record check fails. The code in Listing 4 verified the length of the domain value.

So, is it good? You decide. But, it would be nice to test the logic to ensure that it at least is correct. Listing 8 contains a series of e-mail address test cases that any e-mail validation should pass.

Be sure to run the test to see the valid and rejected e-mail addresses, the double-escaping (\\) inside the PHP strings tends to obfuscate the addresses. You're challenged to subject your favorite e-mail validation code to this test. Be assured that the code in Listing 9 does pass!

Listing 9 contains a complete function for validating an e-mail address. It isn't as concise as many—it certainly isn't a one-liner. But, it is straightforward to read and comprehend, and it correctly accepts and rejects e-mail addresses that many other published functions incorrectly reject and accept. The function orders the validation tests roughly according to increasing cost. In particular, the more complex regular expression and, certainly, the DNS lookup, both come last.

Spread the word! There is some danger that common usage and widespread sloppy coding will establish a de facto standard for e-mail addresses that is more restrictive than the recorded formal standard. If you want to fool the spambots, adopt an e-mail address like, {^c\@**Dog^}@cartoon.com. Unfortunately, you might fool some legitimate e-commerce sites as well. Which do you suppose will adapt more quickly?■

Douglas Lovell is a software engineer with IBM Research, author of *The XSL Formatting Objects Developer's Handbook* published by Sams, and Web site editor for iac52.org.

2007 USENIX ANNUAL TECHNICAL CONFERENCE

Santa Clara, CA • June 17–22, 2007

USENIX

Don't miss the latest in groundbreaking research and cutting-edge practices in a wide variety of technologies and environments.

6 days of training by industry experts, including:

- Richard Bejtlich on TCP/IP Weapons School, Layers 2–3
- Tom Christiansen on Advanced Perl Programming
- Jacob Farmer on Next Generation Storage Networking
- Steve VanDevender on High-Capacity Email System Design
- And over 30 other full- and half-day tutorials

3-day technical program, including:

- The latest research in the Refereed Papers Track
- Keynote Address by Mendel Rosenblum, *Stanford University*
- Expert-led Invited Talks
- Guru Is In Sessions
- BoFs, a Poster Session, and more

Join us in Santa Clara, CA, June 17–22, for the 2007 USENIX Annual Technical Conference.

New in 2007: SANS
SANS Security Training

USENIX '07

Register by June 1 and save! • www.usenix.org/usenix07/lj

CHRISTOF WITTIG and TED NEWARD on OBJECT-ORIENTED LANGUAGE MAPPING to DATABASES

THE PROBLEM OF LANGUAGE-TO-DATABASE MAPPING.

Nicholas Petreley

I urge our readers to have a look at the "Vietnam of Computer Science" by Ted Neward, which compares the quagmire of the Vietnam War to the current quagmire that results from our attempts to blend object-oriented languages with relational or even object-relational databases. A link to Ted's article can be found at **www.odbms.org/vietnam.html**.

The article discusses a problem called object-relational impedance mismatch. Here's how I'd sum up the problem:

We have two great technologies at our disposal: object-oriented languages and relational databases. Problems occur, however, when you try to blend the two, because neither is designed to work seamlessly with the other. A Query-By-Example style of programming may solve the problem, but this works only for simple database access. Mapping classes to tables may work, but the normalization of databases makes this approach difficult. For example, a "customer" table is not likely to include the city and state where the customer lives. A database administrator (DBA) will likely pull out that data and store it by zip code in another table. An option is simply to pass query strings to the database and sort out the fields manually. This is likely to lead to performance problems and breakage if the data types are not handled properly. To confuse matters more, even if you get everything right, there's always the possibility that a DBA will change the database in such a way that it breaks your code.

Languages like PHP include tricks to help smooth out the language-to-database mapping, but even these tricks can be undermined by certain changes to the database schema. The problem lies in the fact that these options are merely tricks, not true database-to-language mapping techniques.

LJ: Does that sum up things fairly well or is there something you want to add?

TED: There's a lot of issues at stake here, one of which is the fundamental tension between developers and DBAs over "who owns the data", and more important, the schema corresponding to that data. Developers who own the schema will create something that's comfortable for them to use, but awkward for the DBAs to manipulate, and DBAs who own the schema will create something that's easy for them to maintain and report from, but which in turn creates awkwardness for the developers. To suggest that a technology—any technology—is going to eliminate completely those problems that are fundamentally rooted in politics is somewhat foolish.

CHRISTOF: The object/relational, dual-schema approach, which puts data above the application and into the hands of a DBA, has organizational advantages in traditional, centralized enterprise IT environments. There is, however, a whole class of applications, where the data store is entirely embedded and invisible to the end user—for example, in device software, in packaged software on your cell phone or PC, in real-time control systems and in SOA applications. These zero-admin database scenarios have no benefit from having two schemata, but incur all the cost of reconciling these inherently incompatible models. Distributed and mobile software architectures, proliferated in a networked world, drive demand for zero-administration database engines, so we will hear a lot more of them in the future.

LJ: I realize that not all databases that call themselves "relational" actually conform to the Codd and Date academic view of relational databases. But, that aside, is this problem limited to relational database mapping to OO languages? Why don't object-relational databases (ORDBMSes) solve this problem? Or do they?

TED: In a lot of ways, the impedance mismatch isn't limited only to objects and relations; trying to stuff objects into a hierarchical data format (like XML) suffers from some of the same problems. So, to suggest that this is "just" an object-relational problem is misleading. Date's position on this is interesting; he insists that an ORDBMS really doesn't exist, that the

fields of an object are, in fact, simply nothing more than columns in a table (or attributes in a relation, to use his more formal terminology). That also implies that inheritance is nothing more than a simple association between tables, somehow silently joined together in a manner that he doesn't seem to specify clearly (at least, not in his eighth edition of *Introduction to Database Systems*). While I'm not going to try to debate relational theory with him, I suspect that his views of the object world are somewhat skewed and therefore not entirely accurate.

LJ: Some have questioned why you drew the comparison between this technical problem and the Vietnam War, and in fact suggested that it's an entirely inappropriate comparison. What are your thoughts on that?

TED: I have two reactions, really. First of all, to all those people who are offended that someone would draw an analogy to Vietnam that wasn't somehow rooted in war or political conflict, I'm sorry that they're offended, but American involvement in Vietnam ended more than 30 years ago, and it's time that we as a nation grow up and stop nursing old wounds. Yes, bad things happened, and some of them happened to us, and some of them happened because of us. It's high time we start looking at Vietnam critically, instead of in a knee-jerk emotional reaction state.

Second, Vietnam is in many ways a perfect analogy to what goes on with many object/relational-mapping tools, not just because Vietnam is the synonym to "quagmire" these days, but because, according to Robert McNamara's recently published memoirs, American leadership knew that they were getting into a potential quagmire, and thought they could manage it somehow. To many, Vietnam was the definition of unclear goals, but McNamara's memoirs make it clear that America was, in fact, trying to "win" the war, which meant winning "the hearts and minds" of the Vietnamese people. It just wasn't clear how they could accomplish that with the tools available to them. O/R-M is a similar situation: it's clear what we want to have happen, it's just not clear how we can make it work.

LJ: I got the impression last time we spoke that db4o users were well aware of this impedance mismatch, and that's why they contributed code and requests that addressed this very problem rather than pressure you to embrace the SQL model. Do I read that right?

CHRISTOF: Yes, what happened was that some managers wanted to see a check mark next to "SQL" in their evaluation spreadsheets. However, OO developers don't want SQL to access their data, unless they have to or are unaware of the alternatives. In fact, SQL is a DBA language, not a developer language. Our developers speak Java and .NET. So, db4o has "Native Queries", the ability to query the database with native Java or .NET semantics, a type-safe and 100% OO approach, for instance.

LJ: And what about reporting, for instance?

CHRISTOF: If you run a distributed application with db4o, you usually don't need reporting (do you run Crystal Reports in your car?). If you still need to link your data at some stage to your back-end RDBMS, then you can use the db4o Replication System (dRS), which uses Hibernate to sync persisted objects into a central relational data warehouse for analysis, backup and so on.

Reporting, specifically, may actually go OO. Several vendors in the Java space (Actuate, Elixir, JasperSoft) and Microsoft in .NET (Visual Studio 2005—ReportViewer) have brought OO reporting tools to the market. And, people may find it easier to report against a plain business object, say "customer", rather than umpteen normalized tables with cryptic names.

LJ: How exactly does db4o address this impedance mismatch?

TED: The db4o approach, like other OODBMSes, avoids the impedance mismatch because we're not trying to store anything other than objects into the system. In other words, there's no "mapping", per se, because there's nothing to map to. (Obviously, internally db4o is doing some storage tricks to avoid blatant inefficiencies, but these are the same tricks that any relational database plays and are, for the most part, entirely black box and removed from the end user's perspective.) This means that the schema of the stored data is that of the objects themselves, thus avoiding the "dual-schema problem" I mentioned in the Vietnam essay.

CHRISTOF: Class model == database schema.

LJ: Is it fair to say that you want db4o to appear as an extension of Java, thus avoiding an impedance mismatch? Is that even possible to accomplish?

TED: I'm not sure I'd say that it's an extension of Java, so much as a mostly transparent persistence system. There have been numerous research projects over the years that have tried to make the persistence entirely transparent, including several within the Java space, and they play interesting tricks like hooking constructors to create persistent objects on "new" calls, and loading objects out of persistent space when invoking non-default constructors, and so on. Most of these haven't made it out of the research space, for a variety of reasons, so I'd be a bit wary of suggesting that db4o "extends" Java (or .NET, for that matter).

CHRISTOF: Strictly speaking, you're right, Ted. But if we have a design philosophy, it is exactly that—let's be as transparent as possible. Let's use the semantics and behavior of Java or .NET for persistence wherever we can to make it least intrusive and most intuitive to developers.

LJ: So the learning curve for db4o is not very steep?

CHRISTOF: No. In fact, there is a podcast on **odbmsjournal.org** (Episode 2) that shows that you are up and running with db4o in five minutes—including the download! What's more challenging though, is that some people have to unlearn bad (=non-OO) habits. They ask, "Where's my primary key?" (There is no primary key in OOP.) So, for us, it is actually easiest to work with young developers, especially in Asia, who have no mental legacy and enjoy a ten-times higher productivity when writing their persistence-related code.

LJ: When I talk to database designers and programmers, I often hear them sing the praises of multi-value databases like PICK. Do you get any demand from your users and developers for multi-value fields?

TED: My experience has been different—that multi-value databases are awkward and difficult to work with. I think what ultimately drives the discussion is what one's own experience is like, and what you find to be obvious and intuitive to you.

For myself, and I think Date would agree with this, multi-value fields are anathema and something to be avoided, because I personally believe pretty strongly in the power of the relational model for data storage and manipulation.

LJ: How does that fit in with O/R mapping? Does that present more problems?
TED: In some respects, no, because it would be "just" a List (or other collection) stored as an attribute of the class. But in other ways, it would represent a significant problem for O/R-M tools, because now trying to decide if a List inside a class should map to a multi-value field or an association to another table would require yet another annotation/attribute on the field to control the mapping, creating even more coupling between the object model and the database schema.

LJ: Does the GPL-ization of Java factor into your work in any way? Does it present problems or opportunities for your business?
CHRISTOF: The GPL-ization, in fact, the dual licensing of Java, is great news for db4o on three accounts. First, open-sourcing Java certainly fosters the Java ecosystem, from IBM to Eclipse to many open-source projects and startups. Second, with open source, we can look to build a much closer integration of db4o's persistence solution with the Java VM than previously possible. Third, the dual-license model itself, as used by MySQL, db4objects, Trolltech and many others, has received further endorsement as a viable open-source business model, which also makes our life as a company much easier.

LJ: How have your customers reacted to the fact that Java is going GPL? Do you see any increase or decrease in interest in Java? A decrease or increase in interest in db4o?
CHRISTOF: We have seen a huge increase in Java users of db4o during the last few months, both in absolute terms as well as in proportion to our other platform, .NET. We don't really know whether this is going back to open-sourcing Java, but that would provide a good explanation.

LJ: Thank you so much for taking the time and effort to speak with us. ■

Nicholas Petreley is Editor in Chief of *Linux Journal* and a former programmer, teacher, analyst and consultant who has been working with and writing about Linux for more than ten years.

An Introduction to Metaprogramming

How to write programs that write programs.

ARIEL ORTIZ

A metaprogram is a program that generates other programs or program parts. Hence, metaprogramming means writing metaprograms. Many useful metaprograms are available for Linux; the most common ones include compilers (GCC or FORTRAN 77), interpreters (Perl or Ruby), parser generators (Bison), assemblers (AS or NASM) and preprocessors (CPP or M4). Typically, you use a metaprogram to eliminate or reduce a tedious or error-prone programming task. So, for example, instead of writing a machine code program by hand, you would use a high-level language, such as C, and then let the C compiler do the translation to the equivalent low-level machine instructions.

Metaprogramming at first may seem to be an advanced topic, suitable only for programming language gurus, but it's not really that difficult once you know how to use the adequate tools.

Source Code Generation

In order to present a very simple example of metaprogramming, let's assume the following totally fictional situation.

Erika is a very smart first-year undergraduate computer science student. She already knows several programming languages, including C and Ruby. During her introductory programming class, Professor Gomez, the course instructor, caught her chatting on her laptop computer. As punishment, he demanded Erika write a C program that printed the following 1,000 lines of text:

```
1. I must not chat in class.
2. I must not chat in class.
...
999. I must not chat in class.
1000. I must not chat in class.
```

An additional imposed restriction was that the program could not use any kind of loop or goto instruction. It should contain only one big main function with 1,000 printf instructions—something like this:

```
#include <stdio.h>
int main(void) {
    printf("1. I must not chat in class.\n");
    printf("2. I must not chat in class.\n");

    /* 996 printf instructions omitted. */

    printf("999. I must not chat in class.\n");
    printf("1000. I must not chat in class.\n");
    return 0;
}
```

Professor Gomez wasn't too naïve, so he basically expected Erika to write the printf instruction once, copy it to the clipboard, do 999 pastes, and manually change the numbers. He expected that even this amount of irksome and repetitive work would be enough to teach her a lesson. But, Erika immediately saw an easy way out—metaprogramming. Instead of writing this program by hand, why not write another program that writes this program automatically for her? So, she wrote the following Ruby script:

```
File.open('punishment.c', 'w') do |output|
  output.puts '#include <stdio.h>'
  output.puts 'int main(void) {'
  1.upto(1000) do |i|
    output.puts "    printf(\"#{i}. " +
      "I must not chat in class.\\n\");"
  end
  output.puts '    return 0;'
  output.puts '}'
end
```

This code creates a file called punishment.c with the expected 1,000+ lines of C source code.

Although this example might seem a bit fabricated, it illustrates how easy it is to write a program that produces the source of another program. This technique can be used in more realistic settings. Let's say that you have a C program that needs to include a PNG image, but for some reason, the deployment platform can accept one file only, the executable file. Thus, the data that conforms the PNG file data has to be integrated within the program code itself. To achieve this, we can read the PNG file beforehand and generate the C source text for an array declaration, initialized with the corresponding data as literal values. This Ruby script does exactly that:

```
INPUT_FILE_NAME = 'ljlogo.png'
OUTPUT_FILE_NAME = 'ljlogo.h'
DATA_VARIABLE_NAME = 'ljlogo'

File.open(INPUT_FILE_NAME, 'r') do |input|
  File.open(OUTPUT_FILE_NAME, 'w') do |output|
    output.print "unsigned char #{DATA_VARIABLE_NAME}[] = {"
    data = input.read.unpack('C*')
    data.length.times do |i|
      if i % 8 == 0
        output.print "\n    "
      end
      output.print '0x%02X' % data[i]
      output.print ', ' if i < data.length - 1
    end
```

```
    output.puts "\n};"
  end
end
```

This script reads the file called ljlogo.png and creates a new output file called ljlogo.h. First, it writes the declaration of the variable ljlogo as an array of unsigned characters. Next, it reads the whole input file at once and unpacks every single input character as an unsigned byte. Then, it writes each of the input bytes as two-digit hexadecimal numbers in groups of eight elements per line. As should be expected, individual elements are terminated with commas, except the last one. Finally, the script writes the closing brace and semicolon. Here is a possible output file sample:

```
unsigned char ljlogo[] = {
    0x89, 0x50, 0x4E, 0x47, 0x0D, 0x0A, 0x1A, 0x0A,
    0x00, 0x00, 0x00, 0x0D, 0x49, 0x48, 0x44, 0x52,

    /* A few hundred lines omitted. */

    0x0B, 0x13, 0x00, 0x00, 0x00, 0x00, 0x49, 0x45,
    0x4E, 0x44, 0xAE, 0x42, 0x60, 0x82
};
```

The following C program demonstrates how you could use the generated code as an ordinary C header file. It's important to note that the PNG file data will be stored in memory when the program itself is loaded:

```
#include <stdio.h>
#include "ljlogo.h"

/* Prints the contents of the array ljlogo as
   hexadecimal byte values. */
int main(void) {
    int i;
    for (i = 0; i < sizeof(ljlogo); i++) {
        printf("%X ", ljlogo[i]);
    }
    return 0;
}
```

You also can have a program that both generates source code and executes it on the spot. Some languages have a facility called eval, which allows you to translate and execute a piece of source code contained within a string of characters at runtime. This feature is usually available in interpreted languages, such as Lisp, Perl, Ruby, Python and JavaScript. In this Ruby code:

```
x = 3
s = 'x + 1'
puts eval(s)
```

The string 'x + 1' is translated and executed when the code is run, printing 4 as a result. Note that even the value bound to variable x is available during the runtime evaluation.

The following Ruby code demonstrates a contrived way to find the result of adding all the integer numbers between 1 and 100. Instead of

using a normal loop or iteration method, we generate a big string containing the expression "1+2+3+...+99+100" and then proceed to evaluate it:

```
puts eval((1..100).to_a.join('+'))
```

The eval function should be used with care. If the string used as the argument to eval comes from an untrusted source (for example, from user input), it can be potentially dangerous (imagine what could happen if the string to evaluate contains the Ruby expression `rm -r *`). In many cases, there are alternatives to eval that are more flexible, less insecure and do not require the speed hit of parsing code during runtime.

Quines

A quine is special kind of source code generator. The jargon file defines a quine as "a program that generates a copy of its own source text as its complete output". You might be right if you think this lacks any practical value by itself, but as a brain-teaser, it can be mind-blowing. Here's a quine written by Ryan Davis, which is one of the shortest ones for the Ruby language:

```
f="f=%p;puts f%%f";puts f%f
```

Run this program, and you will get it as output. You might even try something like this from a shell prompt:

```
ruby -e 'f="f=%p;puts f%%f";puts f%f' | ruby
```

Here we're using the -e option from the command line to specify one line of Ruby source to execute, and then we use a pipe to send its output to another instance of the Ruby interpreter. The output is once again the same program source.

Modifying Programs during Runtime

Dynamic languages, such as Ruby, allow you to modify different parts of your program easily during runtime without having to generate source code explicitly as we did previously. Ruby's core API and frameworks, such as Ruby on Rails, employ this facility to automate common programming tasks. For example, in a class definition, you can use the attr_accessor method to produce the read/write access methods automatically for a given attribute name. Thus, the following code:

```
class Person
  attr_accessor :name
end
```

is equivalent to this more verbose code:

```
class Person
  def name
    @name
  end
  def name=(new_name)
    @name = new_name
  end
end
```

The previous code has a minor drawback: the corresponding instance variable @name is not really created until you first set its value. This means you'll get a nil value if you happen to read the name attribute before writing to it. If you're not careful, this could introduce a few subtle bugs into your programs. The easiest way to avoid this problem is to set the @name instance variable to a reasonable value in the Person#initialize method. Because this is a quite common scenario, wouldn't it be nice to have this method generated automatically, in addition to the read/write accessors? Let's define an attr_initialize method that'll do that using Ruby's metaprogramming facilities.

First, let's briefly address two methods that are key to performing our desired metaprogramming magic:

```
cls.define_method(name) { body }
```

This adds a new instance method to the receiving class. It takes as input the method's name (as a symbol or string) and its body (as a code block):

```
obj.instance_variable_set(name, value)
```

The above code binds an instance variable to the specified value. The name of the instance variable should be a symbol or string, and it also should include the @ prefix.

Now, we're ready to define the attr_initialize class method as an extension to the Object class so that any other class can use it:

```
require 'generator'

class Object
  def Object.attr_initialize(*attrs)
    define_method(:initialize) do |*args|
      if attrs.length != args.length
        raise ArgumentError,
          "wrong number of arguments " +
          "(#{args.length} for #{attrs.length})"
      end
      SyncEnumerator.new(attrs, args).each do
        |attr, arg|
        instance_variable_set("@#{attr}", arg)
      end
    end
    attr_accessor *attrs
  end
end
```

The attr_initialize method takes as input a variable number of attribute names (attrs). Each attribute name has the same position reserved for it in the dynamically created initialize method parameter list (args) in order to set its initial value. We start the new method's code by checking that the number of arguments being received are the same as the number of attributes we originally specified. If not, we raise an error with a descriptive message. Afterward, we use a SyncEnumerator object (from the generator library) to iterate at the same time over the declared attributes list (attrs) and the actual arguments list (args) so as to perform a one-by-one attribute-argument binding using the instance_variable_set method. Finally, we delegate

to the attr_accessor method in order to create the read/write access methods for all the declared attributes.

Here's how we can use the attr_initialize method:

```
class Student
  attr_initialize :name, :id, :address
end

s = Student.new('Erika', 123, '13 Fake St')
s.address = '13 Wrong Rd'
puts s.name, s.id, s.address
```

The expected output would be:

```
Erika
123
13 Wrong Rd
```

Conclusion

Once you're familiar with the techniques, metaprogramming is not as complicated as it might sound initially. Metaprogramming allows you to automate error-prone or repetitive programming tasks. You can use it to pre-generate data tables, to generate boilerplate code automatically that can't be abstracted into a function, or even to test your ingenuity on writing self-replicating code.■

Ariel Ortiz is a faculty member at the Computer Science Department of the Tecnolgico de Monterrey, Campus Estado de Mexico. He's been teaching computer programming for almost two decades. He's not too sure what his favorite programming language is, but he thinks it's either Scheme, Python or Ruby. He can be reached at ariel.ortiz@itesm.mx.

"I'd rather write programs that write programs than write programs."—Richard Sites

Resources

The Jargon File: **www.catb.org/esr/jargon**

Ruby Cookbook by Lucas Carlson and Leonard Richardson, published by O'Reilly Media, 2006. Chapter 10 of this book contains 16 recipes on reflection and metaprogramming using Ruby. Highly recommended.

The Quine Page: **www.nyx.net/~gthompso/quine.htm**. This Web page contains quines in many different programming languages. It even has quines that work in more than one language.

Read Source Code the HTML Way

Cross-reference and convert source code to HTML for easy viewing.

KAMRAN SOOMRO

Every decent programmer has to study source code at some time or other. Sometimes it's to learn new coding styles. Sometimes it's to get an idea of how something works. Regardless of the reason, no programmer can do without it. Studying the source code of small projects is not difficult. You easily can do without formal methods. However, when you want to study the source code of large projects, keeping track of the various functions, variables and their definitions becomes a huge problem. People are lucky if they can find the source code on-line—doing so means they can use tools to process the source code and help them study it. Such tools give people studying source code advantages and flexibility never before dreamed of. One such tool is the LXR (Linux Cross-Reference) tool.

Developed originally as a testbed application for a general hypertext cross-referencing tool in Norway, its flagship achievement is cross-referencing the Linux kernel source code. The code is available at **lxr.linux.no/source** for browsing. Other projects are at **lxr.mozilla.org/seamonkey**, where the Mozilla source code is available for browsing, and the FreeBSD source code is available at **fxr.watson.org**. LXR gives users the capability to jump to function definitions, search for usages and so forth with only a single click. It also supports indexing of e-mail and hypertext links.

The project is based on stock Web technology, so it can be accessed via any Web browser. On the server side, it was developed using Apache but should work with any Web browser supporting CGI-scripting capabilities. The scripts that actually do all the work were developed using Perl and rely heavily on Perl's powerful regular expression libraries.

Probably the best feature of this software is that it is presented to users in HTML format. Because of the HTML format, it is easy to link various portions of the code to others. It is written in Perl, so theoretically, it can run on any operating system that has a Perl interpreter. What is really great about this tool is that it supports multiple languages. This means it doesn't matter which language your program is written in; you still can use this tool to cross-reference and browse your code.

LXR is actually quite simple and clean. The users use a utility called genxref to generate an index of the complete source code. Once this is done, users access a Perl script called source through a Web browser that reads the index files and generates the HTML for the cross-referenced source code dynamically. Users then can browse the source code as they want.

Installation

Installing and configuring LXR is pretty simple once you know a bit about how it works and what the various configuration options are. First, download the source tarball from **sourceforge.net/projects/lxr**. At the time of this writing, lxr-0.3 is the stable release. Once you have downloaded the tarball, extract it using:

```
bash# tar -xvf lxr-0.3.tar.gz
```

After extracting the source, `cd` to the newly created directory, and open the Makefile for editing with the text editor of your choice. You need to set two variables here: INSTALLPREFIX and PERLBIN. PERLBIN refers to the executable binary of the perl5 interpreter. In my case, it was in /usr/bin/perl. INSTALLPREFIX is the directory where LXR will be installed. It should be in a location that is accessible via a Web browser. On my system, that's Apache 1.3.33, and I chose to install it under /var/www/htdcos/. Thus, my Makefile looked something like this:

```
# Makefile for installation and configuration of LXR

# The location of your perl5 binary

PERLBIN=/usr/bin/perl

# LXR will be installed here

INSTALLPREFIX=/var/www/htdocs/lxr

# End of configuration parameters

CGISCRIPTS=find ident search diff source

PERLMODULES=SimpleParse.pm Common.pm Config.pm

....

....

....
```

Leave the rest of the Makefile unchanged. At the console, type:

```
bash# make install
```

and LXR is installed in the specified directory.

Configuration

Now, `cd` to the directory where LXR was installed (in my case that was /var/www/htdocs/lxr). Three subdirectories should be there: bin, http and source. Although the source code you want to cross-reference can be placed anywhere, I prefer to put it under the $INSTALLPREFIX/source subdirectory. I put the glibc-2.3.5 and OpenMOSIX-2.4.26 source code here. Now, we have to generate the index files that LXR will use to generate the cross-referenced source code. So, `cd` to the directory with the source code, and execute the genxref script in $INSTALLPREFIX/bin:

```
bash# /var/www/htdocs/lxr/bin/genxref .
```

The . at the end tells the script that the source code is contained within the current directory. Next, sit back and enjoy the ride until the parsing is complete. Once it's done, you should have two new files in the current directory—the directory containing your source code—fileidx and xref. These two files are the ones lxr needs to generate the cross-referenced source code when you browse it. Make sure that others have read permission for these files. To do so, type the following while still in the source directory:

```
bash# ls -l fileidx xref
```

The output should be something like:

```
-r--r--r-- 1 nobody root 671744 2006-08-24 05:06 fileidx*

-r--r--r-- 1 nobody root 8425472 2006-08-24 05:06 xref*
```

The third r should be set. If it isn't, you can set it by doing the following:

```
bash# chmod o+x fileidx xref
```

Now, it's time to configure LXR for use. Change directory to $INSTALLPREFIX/http/ (in my case, that is /var/www/htdocs/lxr/http/), and open the lxr.conf file for editing. The lxr.conf file is the most important file you need. It has several different configuration options.

Variables

The first thing you'll see when you open the file is a definition for a variable called v. As with programming languages, you can define your own variables and use them later in the configuration file. Wherever they occur, they will be replaced by whatever value they have. Variable values are referenced by the configuration file by $/variable-name. Variable definitions follow one of two possible formats:

```
variable: /variable-identifier, variable-name,
/(/list-of-values/), /default-value/
```

or:

```
variable: /variable-identifier, variable-name,
/[/file-containing-list-of-values/], /default-value/
```

Here's what the terms stand for:

- variable-identifier: the name the variable will be known as throughout the configuration file.
- variable-name: the actual name of the variable that will be displayed to the user.
- list-of-values: comma-separated list of values to be displayed.
- file-containing: a file that contains a list of possible values.
- list-of-values: the list has each entry on a separate line. The user can select any one of them. The absolute path of the file should be provided.
- default-value: the value that the variable will take on by default. The first value is automatically set if this is not specified.

baseurl

The baseurl is the URL relative to which all of the scripts required by LXR are placed. It should be accessible via a browser. In my configuration, it's http://my-ip/lxr/http/ <http://localhost/lxr/http/>. Make sure to place the / at the end, or the last directory will be ignored.

HTML Headers and Footers

When the HTML for the source is generated, LXR can add headers and footers to the pages. Sample headers and footers are provided in the $INSTALLPREFIX/http/ directory. They're called template-head and template-tail. In addition, you also can change the way files and directories are displayed by LXR by modifying the template-dir file. The locations of these files can be specified by the htmlhead, htmltail and htmldir options in the lxr.conf file.

sourceroot

This option tells LXR where to look for the actual source code. In my case, it's /var/www/htdocs/lxr/source/glibc-2.3.5. If you want to cross-reference multiple projects, all you have to do is create a variable specifying the

location of each of the directories that contain the source code. Then, you can specify the value of the variable as the sourceroot. For example, I set up the sources for glibc-2.3.5 and OpenMOSIX-2.4.26, placing the sources for both of them in /var/www/htdocs/lxr/source in their individual directories with the same names as above. In lxr.conf, I had a line like:

```
variable: s, Source, (glibc-2.3.5, OpenMOSIX-2.4.26)
```

then:

```
sourceroot: /var/www/htdocs/lxr/source/$s
```

Thus, the appropriate source code is automatically selected based on the value of the source variable.

srcrootname

srcrootname specifies the name of the project whose source code is displayed—for example:

```
srcrootname: $s
```

incprefix

This specifies the locations of the header files that are to be included in the project.

dbdir

This is the location of the fileidx and xref files generated by genxref. If you have multiple projects, specify a separate location for each, as follows:

```
dbdir: /var/www/htdocs/lxr/source/$s/
```

These are the only options you need to set when configuring LXR. Additionally, you can specify the location of the glimpse binary using glimpsebin.

glimpsebin

glimpse allows users to search for specific files within the source code and to search for any text within source files. You can obtain the latest version of glimpse from **webglimpse.net/trial/glimpse-latest.tar.gz**. Extract and install it. Once you are done installing glimpse, go to the directory where the source code is installed, such as /var/www/htdocs/lxr/source/glibc-2.3.5, and do the following:

```
bash# glimpseindex -H . .
```

The output should look something like this:

```
This is glimpseindex version 4.18.2, 2006.

Indexing "/var/www/htdocs/lxr/source/glibc-2.3.5" ...

Size of files being indexed = 81711416 B, Total #of files = 10075

Index-directory: "/var/www/htdocs/lxr/source/glibc-2.3.5"

Glimpse-files created here:
```

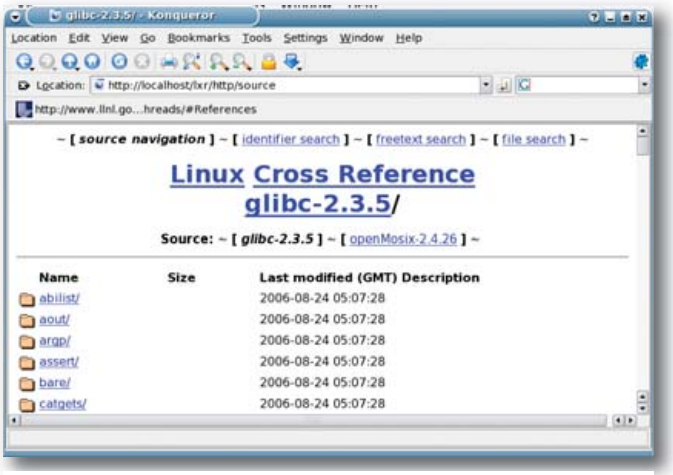


Figure 1. Source Code Navigation via the Browser

```
-rw-r--r-- 1 root root 676398 2006-09-08 05:51 .glimpse_filenames

-rw-r--r-- 1 root root 40300 2006-09-08 05:51 .glimpse_filenames_index

-rw-r--r-- 1 root root 0 2006-09-08 05:50 .glimpse_filetimes

-rw------- 1 root root 1783314 2006-09-08 05:51 .glimpse_index

-rw-r--r-- 1 root root 686 2006-09-08 05:51 .glimpse_messages

-rw------- 1 root root 836 2006-09-08 05:51 .glimpse_partitions

-rw-r--r-- 1 root root 23888 2006-09-08 05:51 .glimpse_statistics
```

This creates the required glimpse index files in the current directory. Once they're created, make sure read permission is set for others:

```
bash# chmod o+r .glimpse-*
```

Now, set the glimpsebin option in lxr.conf to wherever you installed glimpse. I installed it in /usr/local/bin/glimpse.

That's it; save and close the lxr.conf file. The only thing remaining to do now is configure the Web server to work with LXR.

Configuring Apache

The Web server I used was Apache 1.3.33. The first order of business is to set the permissions for the LXR directory. You can do that by editing the httpd.conf file, normally found under /etc/apache/httpd.conf. Add the following lines:

```
<Directory /var/www/htdocs/lxr>

AllowOverride All

Options All

</Directory>
```

Simply replace the location with wherever you've installed LXR.

Then, go to $INSTALLPREFIX/http/ and create and open a file named .htaccess for editing. Type the following lines:

```
<File ~ (search|find|source|diff|ident)?>

SetHandler cgi-script

</Files>
```

This tells the Web server to treat the above-mentioned files as CGI scripts. If you don't do this, the server will display only the contents of these files. Close and save .htaccess. We are now ready to browse the cross-referenced source code.

How to Use LXR

After all of the above steps are done, all you have to do is open a Web browser and go to the URL, http:///my-ip//lxr/http/source, where /my-ip/ is the IP of your Web server. When you open the Web page, you will get something like what is shown in Figure 1.

As you can see, users can select any of the source code files for browsing. At the top, I'm using the template provided by LXR. It includes links to navigate the source code, search for a particular identifier, search for any text within the source code and search for any file.

A few utilities of note:

- Source navigation: select this option to browse the source code of your choice.
- Identifier search: search for the definition and uses of a particular identifier. Requires the ident script.
- Freetext search: search for any text within the source files. Requires the search script.
- File search: search for files matching the passed string. Requires the find script.

All of these utilities also can search using regular expressions to match the strings.

Conclusion

LXR is an excellent tool. It makes life a lot easier for people who want to study source code, and it's powerful and easy to use. The fact that it uses dynamically generated Web pages to browse the source code gives users a lot of flexibility in configuring it. Also, because it can be accessed via any Web browser, it imposes no limitations on the platform, client or location of users. This interoperability is one of the reasons LXR is so powerful.■

Kamran Soomro is a software engineering student at the National University of Sciences and Technology, Pakistan. He has had great interest in Linux since he first used it during his first semester. Since then, he has been avidly involved in promoting Linux in Pakistan.

Answers to the "Where Wants What?" Matchup (from page 16)

1-C, 2-A, 3-Q, 4-L, 5-T, 6-M, 7-N, 8-I, 9-K, 10-P, 11-O, 12-H, 13-G, 14-B, 15-D, 16-R, 17-S, 18-J, 19-F, 20-E

Advertiser Index

Faster Web Applications with SCGI

Speed up your Web applications with SCGI.

JEROEN VERMEULEN

If you're operating a Web server, chances are, you're not merely serving up static text and images. You're likely to be running some Web applications as well, where pages are generated on the fly by some program or script using CGI (Common Gateway Interface). Think of blogging software, bug trackers, news sites and content management systems—anything that turns the browser from a document viewer into a user interface. And, you probably write or at least tweak some of these yourself.

This article shows how to build faster Web applications using an alternative to CGI called SCGI (Simple Common Gateway Interface). SCGI is a protocol, not just a program, but its authors also provide a reference implementation, which is what we use here. It includes modules to use SCGI from Apache or lighttpd and Python classes to help you create SCGI applications. Implementations in other languages are available, but we examine the combination of Apache 2.x and Python here.

Where Does the Time Go?

Normally, a Web application runs briefly, but very frequently, in child processes of the Web server. When a client requests a page, the Web server consults its configuration and finds that the request should go to the application. It delegates the request to a child process, which in turn loads and runs the application program. The program may be a binary or a script in Perl, Python or PHP, shell commands, or just about anything else. The CGI standard defines how the program receives details about the request, including requested URL, requested body, authenticated user identity and originating IP address. The program reads these, produces a page in answer to the client's request, and exits. All this happens again at the next request.

Loading, running and exiting programs can be costly. It does make sense for sloppy programs: they may use memory without ever freeing it up again, for instance. In that case, you want the program to run briefly and then let the operating system clean up after it. But, with today's popular languages—Perl, Python, PHP, Java and shell scripts—there really aren't many problems with this. A well-written application really should be able to handle multiple requests in a single run.

Faster Service with SCGI

SCGI lets your program start once and continue servicing requests for as long as it likes. It works like this: a separate server process, called an SCGI server, runs separately from the Web server and manages one Web application. The Web server forwards all requests for that application to the application's SCGI server. It passes on details about the request in much the same form as in regular CGI.

The SCGI server delegates the request to a child process, just like the Web server did with a regular CGI application. The child process also runs the application, but that's where the similarity ends. Instead of exiting after it's done with that one request, the application can sit and wait for a new one. Each of the SCGI server's child processes runs one instance of the application, each sleeping until there is work for it to do.

The SCGI server spawns a new child process when none are available to take on the latest request—up to a configurable maximum, of course. It also cleans up crashing or exiting child processes, so your Web application can still bail out if things go wrong. But, most of the time, when a request arrives, the application is ready and waiting for it. That's why Ruby on Rails, the Web application framework, comes with the option to run on SCGI; it would be too slow otherwise.

Other Advantages

If the speedup isn't enough for you, there's more. The SCGI server process can be running on the same system as the Web server, but it doesn't have to be. You can offload the server by delegating some Web applications to separate systems, preferably behind a firewall where only the Web server can access them.

Even with just a single server, you can use SCGI to contain vulnerabilities. A normal CGI application starts out running under the same user identity as the Web server process. If an attacker manages to subvert a normal CGI application, your entire Web site may be at risk. An SCGI server, on the other hand, can run under its own user identity, so it can't easily affect the Web server or other applications even if it does run amok. Conversely, you don't need to give the Web server access to the application's code or data anymore; only the application as run by the SCGI server needs access. Everyone else must go through the Web server, which in turn talks to the SCGI server.

You also can run an application in a chroot environment or a virtualized server. With CGI, that quickly becomes expensive and hard to manage. When using SCGI, you start only one server process in your isolated environment—whether it's a chroot jail, a virtualized server, a different user identity or another machine—and the entire application will stay there.

Installing SCGI

You need two components: the Python classes for building SCGI applications and a module for your Web server to make it "speak SCGI" to the applications. If you use Red Hat package management (RPM), you can install these using `yum install python-scgi apache2-mod_scgi`; users of Debian's apt can use `apt-get install python-scgi`

Listing 1. Installing SCGI by Hand

```
# Unpack source directory scgi-1.12 from tarball
tar xzf scgi-1.12.tar.gz
cd scgi-1.12
# Build the Python part
python setup.py build
# Install Python module; we'll need root privileges
sudo python setup.py install
# Now build and install the Apache module
cd apache2
sudo make install
# Enable the SCGI module in Apache. This may fail,
# depending on your Apache version, but no matter.
sudo a2enmod scgi
# Make Apache's new configuration take effect
sudo /etc/init.d/apache2 force-reload
```

`libapache2-mod-scgi`.

You also can install either component by hand. The Apache module requires a C compiler and Apache's apxs script. Some distributions keep apxs in a separate development package rather than installing it as part of the regular Apache package.

Assuming you now have those components, next download the source tarball scgi-1.12.tar.gz, and run the commands shown in Listing 1.

Test Run

Now, let's make sure it all works. The Python package is a module with some classes, and normally, you'd write your application as a program that imports that module. For debugging, however, you also can run it as a standalone application. When it receives a request from the Web server, it simply prints the request's details as a text page. Perfect for a first test—no coding required!

Find the scgi_server.py module on your system. It should be installed in /usr/lib/python2.4/site-packages/scgi (the 2.4 may be 2.3 or 2.5 on your system). Then, run the module:

```
cd /usr/lib/python2.4/site-packages/scgi
python scgi_server.py
```

This listens for requests from the Web server on a TCP port on your system, using port 4000 by default. You can make it listen on a different port by passing the desired port number as a command-line argument, such as:

```
python /usr/lib/python2.4/site-packages/scgi/scgi_server.py 63000
```

The module keeps running until you kill it, so start it in a separate shell. Remember, you don't need to run an SCGI server as root or even under the Web server's identity.

Now that the SCGI application is waiting for requests, pick a location on your Web site to delegate to the application. Let's say you want it to answer all requests for "/scgitest" on this server. Write an Apache configuration snippet, as shown in Listing 2, to a new file in /etc/apache2/conf.d.

Listing 2. Apache Configuration Snippet

```
# Load the SCGI module. This is really only needed
# if you installed manually and the "a2enmod scgi"
# command failed.
LoadModule scgi_module /usr/lib/apache2/modules/mod_scgi.so

<Location "/scgitest">
    # Enable SCGI
    SCGIHandler On
    # Other properties for /scgitest, such as access
    # control
    # ...
</Location>

# Hostname and port number where SCGI server for
# /scgitest is running.
# Port 4000 on localhost (127.0.0.1) is the default.
SCGIMount /scgitest 127.0.0.1:4000
```

Listing 3. scgi_server.py returns request details.

```
SERVER_SOFTWARE: 'Apache'
SCRIPT_NAME: '/scgitest'
REQUEST_METHOD: 'GET'
SERVER_PROTOCOL: 'HTTP/1.1'
QUERY_STRING: ''
CONTENT_LENGTH: '0'
HTTP_ACCEPT_CHARSET: 'UTF-8,*'
HTTP_USER_AGENT: 'Mozilla/5.0'
SERVER_NAME: 'testserver.example.org'
REMOTE_ADDR: '10.99.11.99'
SERVER_PORT: '80'
SERVER_ADDR: '192.0.34.166'
DOCUMENT_ROOT: '/srv/www/'
SERVER_ADMIN: 'webmaster@example.org'
HTTP_HOST: 'testserver.example.org'
REQUEST_URI: '/scgitest'
HTTP_ACCEPT:'text/html,text/plain,*/*;q=0.5'
REMOTE_PORT: '47088'
HTTP_ACCEPT_LANGUAGE: 'en'
SCGI: '1'
HTTP_ACCEPT_ENCODING: 'gzip,deflate'
```

The SCGI server doesn't really need to run on the same machine as the Web server, as you can see here. Simply make sure that the SCGI server's port is properly firewalled, so that only your Web server can reach it! That way, your application can be sure that all CGI parameters have been validated by the Web server first. If an attacker could connect directly to your SCGI application, you wouldn't be able to trust that information. The CGI parameter AUTHENTICATED_USER, for instance, tells your application that the request comes from a particular logged-in user. You can believe that only if you hear it from a

SCGI lets your program start once and continue servicing requests for as long as it likes.

properly configured Web server.

Make Apache reload its configuration with `sudo /etc/init.d/apache2 reload`. Your server should now serve a new location, /scgitest, that simply prints your request's CGI parameters when you access it. Verify this by looking it up in a browser. If your server's address is example.org, point your browser at http://example.org/scgitest. You should see a page that looks like Listing 3.

If that's not what you see, take a look at the shell where you ran the module. It may have printed some helpful error message there. Or, if there is no reaction from the SCGI server whatsoever, the request may not have reached it in the first place; check the Apache error log.

Once you have this running, congratulations—the worst is behind you. Stop your SCGI server process so it doesn't interfere with what we're going to do next.

Writing an Application

Now, let's write a simple SCGI application in Python—one that prints the time.

We import the SCGI Python modules, then write our application as a handler for SCGI requests coming in through the Web server. The handler takes the form of a class that we derive from SCGIHandler. Call me unimaginative, but I've called the example handler class TimeHandler. We'll fill in the actual code in a moment, but begin with this skeleton:

```
#! /usr/bin/python
import scgi
import scgi.scgi_server

class TimeHandler(scgi.scgi_server.SCGIHandler):
    pass  # (no code here yet)

# Main program: create an SCGIServer object to
# listen on port 4000.  We tell the SCGIServer the
# handler class that implements our application.
server = scgi.scgi_server.SCGIServer(
    handler_class=TimeHandler,
    port=4000
    )
# Tell our SCGIServer to start servicing requests.
# This loops forever.
server.serve()
```

You may think it strange that we must pass the SCGIServer our handler class, rather than a handler object. The reason is that server object will create handler objects of our given class as needed.

This first incarnation of TimeHandler is still essentially the same as the original SCGIHandler, so all it does is print out request parameters. To see this in action, try running this program and opening the scgitest page in your browser as before. You should see something like Listing 3 again.

Now, we want to print the time in a form that a browser will understand. We can't simply start sending text or HTML; we first must emit an HTTP header that tells the browser what kind of output to expect. In this case, let's stick with simple text. Add the following near the top of your program, right above the TimeHandler class definition:

```
import time
def print_time(outfile):
    # HTTP header describing the page we're about
    # to produce. Must end with double MS-DOS-style
    # "CR/LF" end-of-line sequence. In Python, that
    # translates to "\r\n.
    outfile.write("Content-Type: text/plain\r\n\r\n")

    # Now write our page: the time, in plain text
    outfile.write(time.ctime() + "\n")
```

By now, you're probably wondering how we will make our handler class call this function. With SCGI 1.12 or newer, it's easy. We can write a method TimeHandler.produce() to override SCGIHandler's default action:

```
class TimeHandler(scgi.scgi_server.SCGIHandler):
    # (remove the "pass" statement--we've got real
    # code here now)

    # This is where we receive requests:
    def produce(self, env, bodysize, input, output):
        # Do our work: write page with the time to output
        print_time(output)
```

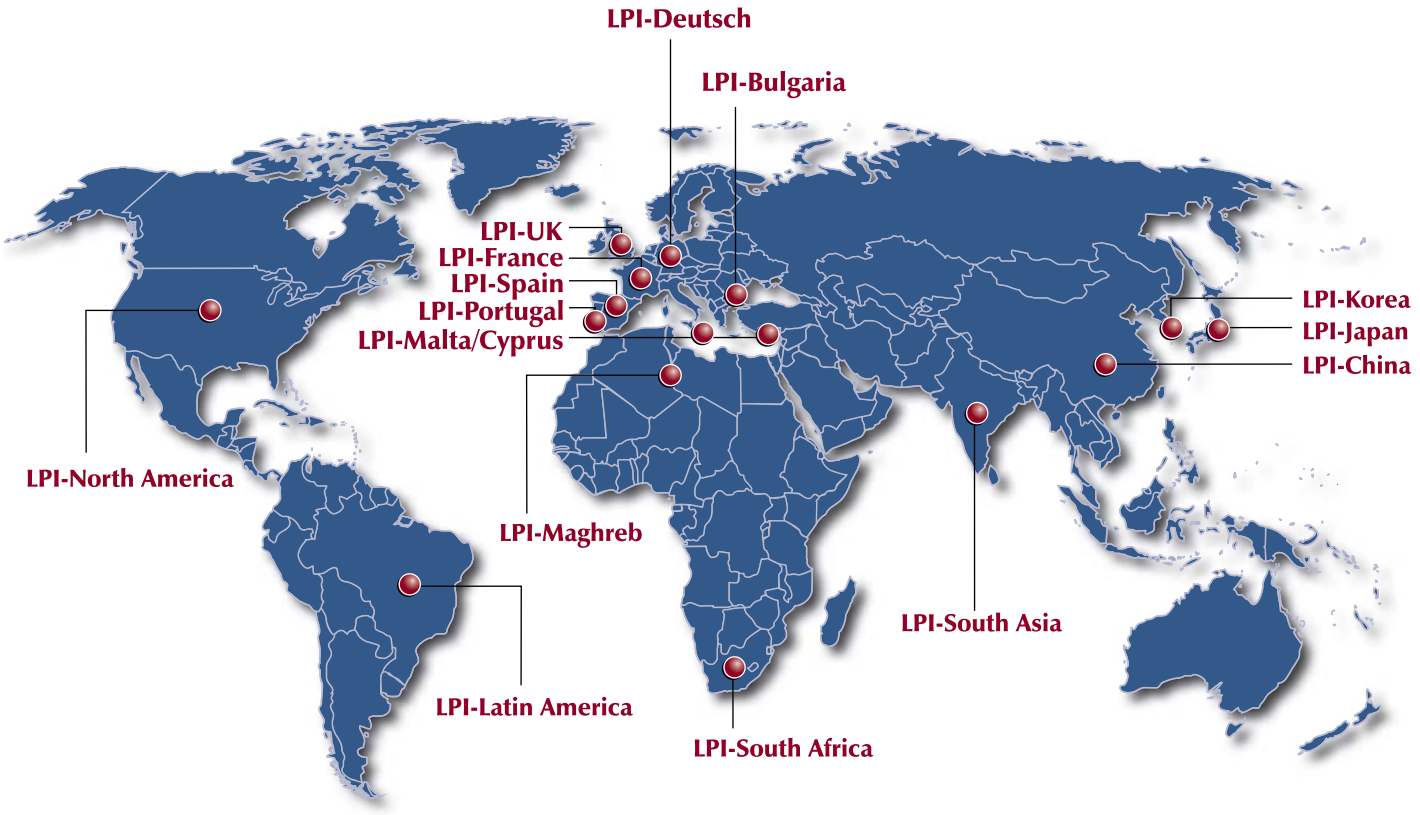
We ignore them here, but produce() takes several arguments: env is a dict mapping CGI parameter names to their values. Next, bodysize is the size in bytes of the request body or payload. If you're interested in the request body, read up to bodysize bytes from the following argument, input. Finally, output is the file that we write our output page to.

If you have SCGI 1.11 or older, you need some wrapper code to make this work. In these older versions, you override a different method, SCGIHandler.handle_connection(), and do more of the work yourself. Simply copy the boilerplate code from Listing 4 into the TimeHandler class. It will set things up right and call produce(), so nothing else changes, and we can write produce() exactly as if we had a newer version of SCGI.

Once again, run the application and check that it shows the time in your browser.

Next, to make things more interesting, let's pass some arguments to the request and have the program process them. The convention for arguments to Web applications is to tack a question mark onto the URL, followed by a series of arguments separated by ampersands. Each argument is of the form name=value. If we wanted to pass the program a parameter called pizza with the value hawaii, and another one called drink with the value beer, our URL would look something like http://example.org/scgitest?pizza=hawaii&drink=beer.

Growing a World of Linux Professionals



We at the Linux Professional Institute believe the best way to spread the adoption of Linux and Open Source software is to grow a world wide supply of talented, qualified and accredited IT professionals.

We realize the importance of providing a global standard of measurement. To assist in this effort, we are launching a Regional Enablement Initiative to ensure we understand, nurture and support the needs of the enterprise, governments, educational institutions and individual contributors around the globe.

We can only achieve this through a network of local "on the ground" partner organizations. Partners who know the sector and understand the needs of the IT work force. Through this active policy of Regional Enablement we are seeking local partners and assisting them in their efforts to promote Linux and Open Source professionalism.

We encourage you to contact our new regional partners listed above.

Together we are growing a world of Linux Professionals.

Linux Professional Institute

Stable. Innovative. Growing.

Listing 4. Boilerplate Code for SCGI 1.11 or Older

```
# Insert this definition into your handler class:
class TimeHandler(scgi.scgi_server.SCGIHandler):

    # ...

    def handle_connection(self, conn):
        input = conn.makefile("r")
        output = conn.makefile("w")
        env = self.read_env(input)
        bodysize = int(env.get('CONTENT_LENGTH',0))
        try:
            self.produce(env,bodysize,input,output)
        finally:
            output.close()
            input.close()
            conn.close()
```

Any arguments that the visitor passes to the program end up in the single CGI parameter QUERY_STRING. In this case, the parameter would read "pizza=hawaii&drink=beer". Here's something our TimeHandler might do with that:

```
class TimeHandler(scgi.scgi_server.SCGIHandler):
    def produce(self, env, bodysize, input, output)
        # Read arguments
        argstring = env['QUERY_STRING']
        # Break argument string into list of
        # pairs like "name=value"
        arglist = argstring.split('&')

        # Set up dictionary mapping argument names
        # to values
        args = {}
        for arg in arglist:
          (key, value) = arg.split('=')
          args[key] = value

        # Print time, as before, but with a bit of
        # extra advice
        print_time(output)
        output.write(
         "Time for a pizza. I'll have the %s and a swig of %s!\n" %
         (args['pizza'], args['drink'])
        )
```

Now the application we wrote will not only print the time, but also suggest a pizza and drink as passed in the URL. Try it! You also can experiment with the other CGI parameters in Listing 3 to find more things your SCGI applications can do.

Even with just a single server, you can use SCGI to contain vulnerabilities.

Porting Applications

Once you're comfortable writing programs using SCGI, you may want to try adapting existing applications to use it. Some well-known Web applications, such as MoinMoin (a wiki) and Trac (a wiki-based collaborative development environment), are implemented as Python modules. Both of these examples come with CGI scripts in Python that can be called from Apache. The CGI scripts are very short; they really don't do anything except import the application's modules and invoke a function on them.

If you find an application like that, all you really need to do to make it work with SCGI is take that little bit of Python code and move it into a produce() method, as in the examples you've seen here. If you have SCGI 1.12 or newer, you also might want to take a look at an alternative SCGIHandler method, produce_cgilike().

Conclusion

That's about all we have room for. If you wonder about how the CGI parameters work, try looking at the CGI standard, which calls them "request meta-variables" (see Resources).

Finally, a word of warning. You'll notice that the last example program dies horribly if you fail to pass the expected arguments. The SCGI server replaces the failing processes, so in this case, there's no real problem. But, this should remind you how careful you need to be when writing Web applications. Never trust the input you receive from outside! If a program can be crashed, someone can probably subvert it or take it out of action. People all over the world do that sort of thing for fun or profit, so take the risk seriously.■

Jeroen Vermeulen works for the Open Source Department of the Thai Software Industry Promotion Agency. He's currently working on Suriyan, a server system for those who don't have time for server systems.

Resources

SCGI Downloads: **quixote.python.ca/releases**

SCGI Home Page: **www.mems-exchange.org/software/scgi**

CGI Standard: **ftp.rfc-editor.org/in-notes/rfc3875.txt**

More on SCGI with Python and Apache2: **thaiopensource.org/development/suriyan/wiki/UsingScgi**

Perl Interface: **search.cpan.org/~vipercode/SCGI/lib/SCGI.pm**

Lisp Interface: **randallsquared.com/download/scgi**

Trac: **trac.edgewall.com**

MoinMoin: **moinmoin.wikiwikiweb.de**

Extend OpenOffice.org

It's easier than you might think to create your own OpenOffice.org extensions.

DMITRI POPOV

If you have a nifty macro or a nice Writer template you want to share with other OpenOffice.org users, publishing them on the Web along with detailed installation instructions is probably not the best way to go. Fortunately, OpenOffice.org supports extensions—small installable packages that provide added functionality. You easily can turn your templates, autotext entries, gallery art and macros into extensions that can be installed with a couple of clicks. Better yet, OpenOffice.org's extensions have an easy-to-understand and well-defined architecture, and you can start building your own extensions in no time.

Extending OpenOffice.org's functionality using extensions is nothing new. From the very beginning, users could add new features to the office suite by installing so-called UNO packages. Usually, these packages contained OOo Basic code, and they offered a more straightforward way of integrating macros into OpenOffice.org applications. With the release of OpenOffice.org 2.0.4, the idea of adding new features via installable packages has been rethought thoroughly and aligned with a concept that is more familiar to end users—namely the extension architecture of the Mozilla Firefox browser.

The technical implementation of the extension system in OpenOffice.org also has been reworked. Most notably, the new version of OpenOffice.org can handle so-called non-code extensions that can contain document templates, gallery items, autotext snippets and so on. The new version of OpenOffice.org also introduces the new .oxt file extension that allows users to identify installable extension packages easily.

How Extensions Work

An OpenOffice.org extension is essentially a zip file that includes both the installable contents and additional metadata required to install and register the extension properly. OpenOffice.org provides a simple-to-use tool called Package Manager that lets users install new extensions and manage existing ones. To install an extension, simply choose Tools→Package Manager, select My Packages, and press the Add button. Once the extension is installed, restart OpenOffice.org, and you are good to go. Unlike Firefox, some types of extensions don't require you to restart OpenOffice.org. For example, if you install non-code extensions, you can use them right away.

To better understand the anatomy of OpenOffice.org extensions, let's dissect an empty sample extension from the OpenOffice.org Wiki (**wiki.services.openoffice.org/wiki/Non-code_extensions**). In order to peek inside the package, you have to change its extension from oxt to zip. This allows you to treat the package as a regular zip archive. The package consists of three elements: the META-INF and template folders, as well as the Paths.xcu configuration file. The META-INF folder contains the manifest.xml file that, among other things, "points" to the Paths.xcu configuration file. The Paths.xcu file contains information that the Package Manager uses

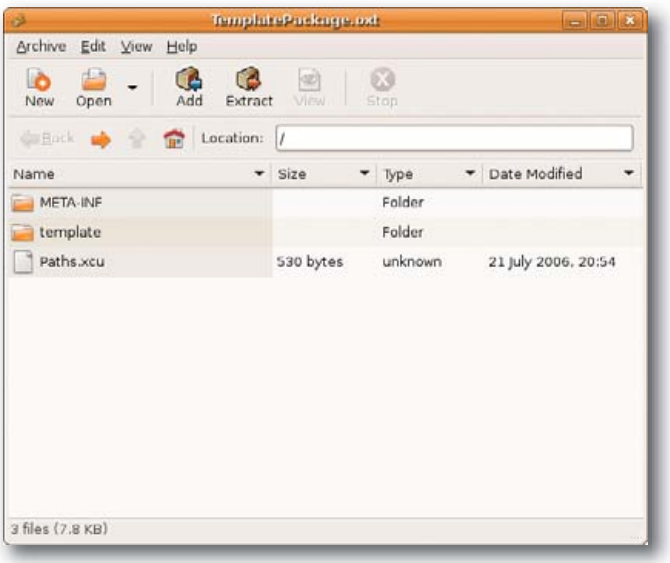


Figure 1. The Contents of a Non-Code OpenOffice.org Extension

to add the templates to the appropriate location. This location is defined as %origin%/template, and the Package Manager replaces the %origin% variable with the full path to the internal template container. The fuse parameter adds the templates to the specified container or creates a new one if it doesn't exist. To create a new template extension, you don't need to tweak anything; the configuration file and the overall structure of the extension remain the same. All you have to do is copy your custom templates into the template folder. Change the file extension of the resulting package back to oxt, and install it via Tools→Package Manager. To check whether the extension has been installed properly, choose File→Templates→Organize; you should see your templates in the My Templates folder.

Creating a Programmatic Extension from Scratch

Although creating non-code extensions is rather trivial, building packages containing code (let's call them programmatic extensions) is a different matter. The programmatic extension includes not only the code itself, but also a more complex configuration file containing information about menus, submenus, commands and macros assigned to them, icons and so forth. Creating a configuration file manually, even for the most simple programmatic extension, requires some technical knowledge, and it can be rather time consuming. Fortunately, there is a tool that can automate the entire process of creating an extension. Although the Add-on Tool (**documentation.openoffice.org/HOW_TO/various_topics/Addons1_1en.sxw**) hasn't been updated since 2003, it still does a great job of generating extensions that can be used with

the latest version of OpenOffice.org. To get to grips with the Add-on Tool and better understand the process of creating a programmatic extension, let's build a simple dummy text-generator extension from scratch. Once installed, the extension adds the Lorem ipsum command to the Tools→Add-Ons menu. This command runs an OpenOffice.org Basic macro that inserts a specified number of paragraphs with the Lorem ipsum dummy text. The following description assumes that you have a general knowledge of how to create and manage macros, modules and libraries in OpenOffice.org. The Add-on Tool uses the older term "add-on", which you can consider a synonym of "extension".

Start with creating a macro that generates the dummy text. To keep things tidy, create a separate library called LoremipsumLib, containing the LoremipsumModule. In this module, add the macro shown in Listing 1. (Replace the "Lorem ipsum dolor sit amet..." string with a paragraph of dummy text).

Before you fire up the Add-on Tool, you need to do some preparatory work. First, create a separate folder for all your working files (for example, loremipsum). If you want to add icons to the menu items, make sure you have the necessary graphics files. According to the official documentation, you need a set of small (16x16) and big (26x26) icons in BMP format. However, you also can use 16x16 icons in PNG format (you can find some high-quality icons

Listing 1. loremipsummacro.txt

```
Option Explicit
Sub LoremipsumMacro()
Dim ThisDoc As Object
Dim Cursor As Object
Dim ParNumber As Integer
Dim InputMsg As String, InputTitle As String, InputReturn As String
ThisDoc=ThisComponent
InputMsg="Number of paragraphs"
InputTitle="Lorem Ipsum Generator"
InputReturn=InputBox (InputMsg, InputTitle)
ParNumber=InputReturn
Do While ParNumber>0
Cursor=ThisDoc.text.createTextCursor
Cursor.String="Lorem ipsum dolor sit amet..." & Chr(13)& Chr(13)
ParNumber=ParNumber-1
Loop
End Sub
```

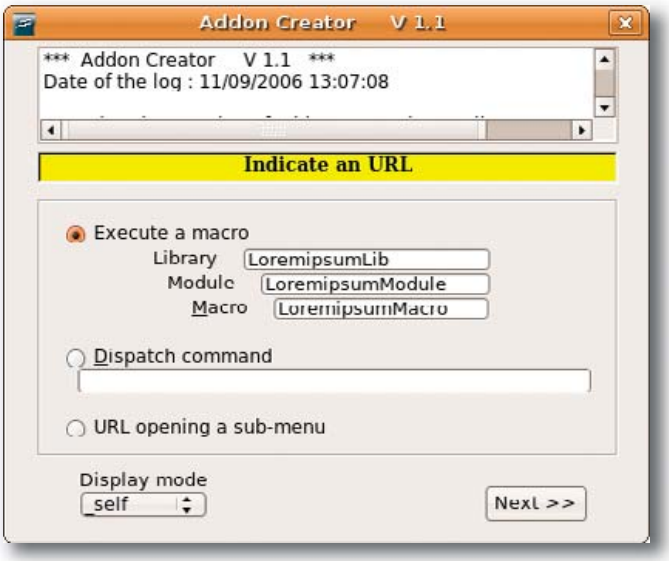


Figure 2. Using the Addon Creator to Create a Programmatic Extension

at **www.famfamfam.com/lab/icons/silk**). Next, copy the entire LoremipsumLib library into the loremipsum folder. To do this, navigate to .OpenOffice.org2/user/basic inside your home directory, and copy LoremipsumLib into the loremipsum folder. Finally, copy the icons into the LoremipsumLib folder. Now, open the Add-on Tool document, and make sure that macro execution is enabled. Scroll to the Create the configuration file chapter, and press the Create XML file button to launch the Addon Creator.

The process of creating an extension using the Addon Creator can be roughly divided into three stages. First, you define the general setting, including the top-level menu and its position. Then, you specify the menu items, and finally, you zip the created package.

In the Basic Information window, specify the path to the main script file. Press the Browse button, and select the script.xlb file inside the LoremipsumLib folder. You also must specify a name for your extension in the Unique name for your addon field. Simply replace the example part in the org.openoffice.Office.addon.example string with the name you want (for example, org.openoffice.Office.addon.Loremipsum). Press Next to choose where to add the top-level extension menu. You have two choices here: you either can add a menu item to the Main menu or under the Tools menu. As a rule of thumb, if you have a simple extension containing only a couple of commands, tuck it under the Tools menu. A more complex extension deserves its own entry in the Main menu. Because the Lorem ipsum generator contains only one command, it makes sense to install it under the Tools menu. Next, enter a menu title, and press the Add this text button. If you want to make your extension available only for a particular language or country, you may do so by specifying the appropriate settings in the Language restrictions section. Press Finished when you are satisfied with the settings.

The next step is to link the LoremipsumMacro to the created menu item. To do this, you have to specify the library, the module and the macro itself. In our case, these are LoremipsumLib, LoremipsumModule and LoremipsumMacro, respectively. Once you have linked the macro to the command, you can add an icon to it.

Listing 2. addonxcu.txt

```
<?xml version='1.0' encoding='UTF-8'?>
<oor:node xmlns:oor="http://openoffice.org/2001/registry"
xmlns:xs="http://www.w3.org/2001/XMLSchema" oor:name="Addons"
oor:package="org.openoffice.Office">
  <node oor:name="AddonUI">
    <node oor:name="AddonMenu">
      <node oor:name="org.openoffice.Office.addon.Loremipsum"
oor:op="replace">
        <prop oor:name="Context" oor:type="xs:string">
          <value/>
        </prop>
        <prop oor:name="Title" oor:type="xs:string">
          <value>Lorem ipsum</value>
        </prop>
        <prop oor:name="URL" oor:type="xs:string">
 <value>macro:///LoremipsumLib.LoremipsumModule.LoremipsumMacro
 </value>
        </prop>
        <prop oor:name="Target" oor:type="xs:string">
          <value>_self</value>
        </prop>
        <prop oor:name="ImageIdentifier" oor:type="xs:string">
          <value/>
        </prop>
      </node>
    </node>
    <node oor:name="Images">
      <node
oor:name="org.openoffice.Office.addon.Loremipsum.img01"
oor:op="replace">
        <prop oor:name="URL" oor:type="xs:string">
 <value>macro:///LoremipsumLib.LoremipsumModule.LoremipsumMacro
 </value>
        </prop>
        <node oor:name="UserDefinedImages">
          <prop oor:name="ImageSmallURL">
            <value>%origin%/LoremipsumLib/Icon.png</value>
          </prop>
        </node>
      </node>
    </node>
  </node>
</oor:node>
```

Because we've chosen to use an icon in PNG format, press the Other image type button, select the 16x16 normal contrast item from the Icon definition drop-down list, select the icon using the Browse button, and press OK to add it. When adding icons, you have two options: you either can link to an icon that will be added to the extension as an image file, or you can integrate it into the configuration file (this works only with icons in BMP format). Which option you choose is more or less a matter of taste, but linking to icons rather than embedding them produces a cleaner

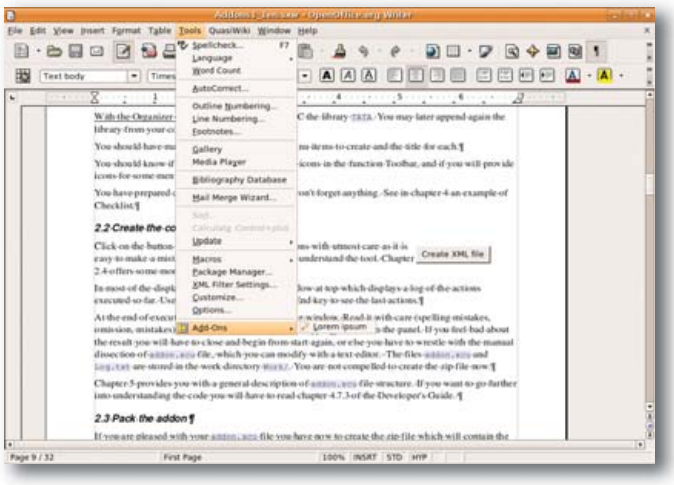


Figure 3. The Lorem Ipsum Generator Extension in Action

and easier-to-read configuration file. This can come in handy if you need to edit the file manually later. Use the Finished button to finalize the extension, and press the Addon zipping button to pack it. Now you can install the created extension by choosing Tools→Package Manager. Restart OpenOffice.org, and you should see the Lorem ipsum command in the Tools→Add-Ons menu.

Tweaking Extensions

The Addon Creator conveniently hides the technical part of the process, which is good if you don't want to spend time doing the donkey work manually. This is, however, less useful if you want to gain a better understanding of what makes extensions tick—not only to satisfy your curiosity, but also to be able to troubleshoot your extensions and tweak them without running the Addon Creator every single time.

If you look inside the zip package, you will notice that it contains the familiar META-INF folder, a folder with the macro files and the addon.xcu file (Listing 2). The latter is the key element of the extension, as it contains all the configuration data. The addon.xcu is based on XML, and even if you have only a basic knowledge of XML, you easily can figure out how it works simply by looking at its contents.

The XML file contains a number of nodes, and each node has properties, which, in turn, have values. For example, the top node <node oor:name="AddonMenu"> has multiple properties, such as <prop oor:name="Title" oor:type="xs:string">, that have a value containing the extension's menu title <value>Lorem ipsum</value>. The <prop oor:name="URL" oor:type="xs:string"> property has the <value>macro:///LoremipsumLib.LoremipsumModule.LoremipsumMacro</value> value, which contains the link to the appropriate macro. Knowing that, you can modify the extension by tweaking its addon.xcu file. For example, if you want to change the menu title, you simply can edit the <value>Lorem ipsum</value> value as follows:

```
<prop oor:name="Title" oor:type="xs:string">
  <value>Insert dummy text</value>
</prop>
```

In more complex macros, you even can add new menus and commands simply by cloning and modifying parts of the configuration file.

Final Word

Now that you know the basics, you can start building your own OpenOffice.org extensions. If you want to share your creations with other users, you can add them to the official extension repository (**wiki.services.openoffice.org/wiki/Extensions_repository**). Most of the extensions there are released under the GPL, so you can dismantle them to see how they work and get new ideas.■

Dmitri Popov is a freelance writer whose articles have appeared in Russian, British and Danish computer magazines. His articles cover open-source software, Linux, Web applications and other computer-related topics.

A Look at Lua

An overview of the compact yet powerful Lua programming language.

JOSEPH QUIGLEY

Lua is a free and open-source multi-paradigm programming language released under the MIT license. Created in 1993 by Roberto Lerusalimschy, Luiz Henrique de Figueiredo and Waldemar Celes, Lua is a dynamically typed language. Extremely compact (only 150KB compiled), it is primarily used as a scripting language or an extension to another language (mainly C/C++).

What Is Lua and How Is It Used?

Lua is implemented as a library and has no "main" program. It works only when embedded in a host client. The interpreter is a C program that uses the Lua library to offer a standalone Lua interpreter. Rather than provide a complex and rigid specification for a single paradigm, Lua is intended to be extended to fit different problem types.

Being small, Lua fits on many host platforms and has been ported and used in video games for both the PlayStation Portable and the Nintendo DS, and it is used in larger games, such as *FarCry* and *World of Warcraft*. The Adobe Photoshop Lightroom photography program and the lighthttpd Web server have incorporated Lua as well. Lua has a few advanced features, primarily coercion, coroutines, garbage collection, first-class functions and dynamic module loading. Because Lua is small, it includes only a few data types. It attempts to maintain a balance between power and small size.

What's Different about Lua?

Lua is comparably as easy as Python in terms of learning how to write code. Of the two, Lua is usually the better choice for embedded systems, simply because it's smaller. Lua's strength is in processing strings and tables. It handles logical equations more adeptly than Python.

For a quick hack, a Lua programmer can process complicated data more quickly and easily than a Python programmer can (although a Ruby programmer can do so almost as quickly). But, for a large application that handles many chunks of complex data, a heavier language such as Ruby or Python may be a better choice.

There is no need to worry about different types of integers. You may have found that the different types of integers and numbers (such as floats, longs or doubles) can screw up the output of your program or even make it crash if you are absent-minded. Lua uses coercion for every integer and number type to convert it into a single type. You can add a float, long integer or a double to any other type of integer or number without a hitch in Lua. In contrast, doing this can cause programs written in Python 2.4 or older versions to crash. Lua is extremely forgiving syntactically. What if, for some reason, you are programming on an embedded device with a four-inch wide screen? You can reduce the amount of lines and other characters, which in turn enables easy reading of the code to make up for the small screen.

Small is beautiful. A programmer can embed Lua into several other languages, such as C/C++ and Java, without bloating the host language, because Lua has a tiny API. Similar to Lisp's single data structure, tables are the only data structuring mechanism that Lua has. This makes tables very powerful, because with a little work, they can emulate data structures in larger languages.

Object-oriented programming implementation is minimalistic. Lua uses tables and functions rather than classes.

In contrast to Python, Lua does not focus on 100% backward compatibility. Many newer releases of Lua break programs written in previous versions. Fortunately, the Lua developers always announce what the new versions of Lua will break.

Lua supports threading. Multiple Lua interpreters can coexist in the same process, and each one can run independently in its own thread. This often makes Lua desirable for multithreaded programs in embedded systems.

Installing Lua

To compile and install Lua from the source code, grab a copy of Lua 5.1 from the Lua.org Web site, and untar, configure, make and install it:

```
tar -xvzf lua-5.1.1.tar.gz
cd lua-5.1.1
make xyz
make xyz install
```

(*xyz* is your platform name.)

Lua should now be installed and working. To test your install, type `lua` on the command line. An interactive interpreter should appear.

Syntax

Lua is a dynamically typed language whose syntax is very similar to that of Python and even more similar to Ruby. Line breaks do not play any role in Lua's syntax, like that of Python or Ruby. Take, for example, the following ugly, but valid code:

```
foo = 89
bar = foo+2
print(bar)
```

Because it is small, Lua has only eight basic data types:

1. nil (similar to Python's None)
2. booleans
3. numbers
4. strings
5. functions
6. userdata (a type that allows arbitrary C data to be stored in Lua variables)
7. threads
8. tables

Lua supports only a few data structures, including arrays, lists and hash tables.

The table type implements an associative array that can be indexed with any value (similar to Python), except nil (dissimilar to Python). Nil's goal is to be different from any other value, as well as the default value for global variables. Nil also plays a much more important role in

Lua than None does in Python. Although tables are the only data structuring mechanism in Lua (which may seem like a disadvantage), the table is just as powerful as Python's dictionary and list, and it's even as powerful as Ruby's hash. Tables are used to represent many different types of arrays, sets, trees and several other data structures. One handy feature of tables is to use strings as keys—for example:

```
x = { ["hello world!"] = "ciao world!" }
print(x["hello world!"])
```

When running this example, Lua outputs "ciao world!" and not "hello world!" as it might appear.

For a more in-depth look at Lua's tables go to **lua-users.org/wiki/TablesTutorial**.

Variables and Identifiers

Because any value can represent a condition, booleans in Lua differ from those in many other languages. Both false and nil are considered false in Lua, but Lua considers everything else true (including zero and an empty string).

Unlike Python, global variables do not need to be declared. To create one, assign a value to it. To delete it, give it the nil value. A global variable exists only if it has a non-nil value. Exactly the opposite of Python, most variables in Lua are global by default, and you must declare the variable "local" to make it a local variable rather than assuming that all variables are local.

Because most CPUs perform floating-point arithmetic just as fast as integer arithmetic, numbers in Lua represent real, double-precision, floating-point numbers rather than common integers. Because Lua doesn't need integer types, it doesn't have them. This eliminates rounding errors, floating-point numbers and long integers.

Lua handles strings very adeptly and has been used for strings that are several megabytes long. It converts between strings and numbers; any numeric operation applied to a string converts the string to a number. This conversion principle applies only to numbers, as Lua converts numbers to strings when strings are expected. Even with automatic conversion, Lua still can differentiate between numbers and strings in cases like 90 == "90" (which always is false).

Identifiers starting with an underscore (such as _FOO) are not recommended for use in Lua, because many are reserved for special uses. As long as an identifier does not begin with a digit, the identifier can be made up of a combination of underscores, letters and digits.

You can basically rename anything in Lua, even to the point of making it un-callable. Take, for example, the following code:

```
x = io
x.read()
io = "Hello world!"
x = "Let's make io uncallable!"
io.read()
```

The second line gets keyboard input through the io module. Because io is essentially a variable with a function as a value, you can give it a different value so that io does not relate to the input/output functions anymore. When you try to get keyboard input from the io module again, Lua returns an error. The program is unable to call the input/output functions now that io's value has been reassigned. In order to use io again, you must restart the Lua program.

Operators and Assignment

Lua concatenates strings with the .. operator. Note that `print("Hello".."World!")` is valid, but `print("I've said 'Hello World'"..5.."or more times.")` is not. This is because Lua sees the periods as decimals after the integer. The operator must have a space between the strings and the integer. Otherwise, it won't return an error. The following code validly concatenates the strings and the integer:

```
print("I've said 'Hello World' " ..5 .. " or more times.")
```

Lua uses many of the common operators that Python, Ruby and most every other language use. For the Python/Ruby logical not operator, Lua can either use it or use ~= for the negation of equality. Always

remember that Lua treats strings and integers differently: `"1" < 2` is always false, and strings are compared alphabetically.

Lua ends while loops if the condition is false. Repeat-until statements are the opposite of while loops; they loop until the condition is true. for loops have some hidden twists, which can be annoying to Python or Ruby programmers. Local variables created in the for loop are visible only inside the loop. The variable does not exist when the loop ends, so if you need the value of the control variable, you have to save its value into another loop. Breaks or returns should appear only as the last statement before an end, an else or an until in a loop for syntactic reasons.

Lua treats functions as "first class" values and uses them for OOP (object-oriented programming). Lua can call its own functions or C functions, and it handles functions as a type. You can give a variable the function property or create it with the function() method. Functions written in Lua can return multiple results if you list them after a return keyword.

Lua supports OOP, but due to Lua's size, its implementation of OOP lacks a few features. Lua uses tables and functions for OOP rather than classes. In the same way that Python accesses a function or variable in a class, Lua accesses it with Table.function or Table.variable.

Lua can be picky when it comes to multiple assignment, because it adjusts the number of values on the assignment. If the amount of values is less than the list of variables, all remaining values are given the nil value. If the list of values is longer than the amount of variables, Lua silently discards them.

Object-Oriented Programming

Lua has some basic OOP capabilities. The self parameter is an integral concept in any object-oriented language, and it is one of the few OOP concepts that Lua has. Many object-oriented languages tend to hide the self mechanism from you so that you do not have to declare this parameter. Lua hides this parameter with the colon operator. You also can use the colon, a function and a table to emulate a class. Because Lua does not have the class concept, each object defines its own behavior and shape:

```
Earth = {martians = 5389}
function Earth:casualties (survivors)
 Earth.martians = Earth.martians - survivors
 print("Earth is free! "..Earth.martians.." martians survived!")
end

Earth:casualties(5380)
```

The colon in the above example is used to add an extra parameter in the method definition. It also adds an extra argument in the method call. You don't have to use the colon. Lua programmers can define a function with the dot syntax and call it with the colon syntax, or vice versa if they add an extra parameter:

```
Earth = {martians = 5389,
 casualties = function (self, survivors)
  self.martians = self.martians - survivors
  print("Earth is free! "..self.martians.." martians survived!")
 end
}

Earth.casualties(Earth, 5380)
Earth.martians = 5389
Earth:casualties(5380)
```

In this case, the function had to be part of the table so that it could be called via the dot or the colon syntax. Note that I also had to give the function the self parameter for either calling method to work. Although these are simple OOP examples that scratch only the surface of OOP, you can find out about inheritance and other OOP implementations in the Lua Reference Manual or in the book *Programming in Lua* (both are available for free from the Lua Web site).

Show and Tell

Now, let's compare programming in Lua to programming in Python. First, let's write a trivia game. Here is some simple Lua code that uses a table as a dictionary to store both the questions and the answers:

```
print("What's your name?")
name = io.read()
questions = {
  ["Which came first? Minix or Unix?"] = "Unix",
  ["Who created Linux?"] = "Linus Torvalds",
  ["In what year was Linux created?"] = "1991"
}

correct_answers = 0
for key,value in pairs(questions) do
print(key)
answer = io.read()
if answer == value then
  correct_answers = correct_answers + 1
end
end
if correct_answers == 0 then
print("You need to browse Wikipedia!")
else
print("\nGood job, "..name.."!")
print("Correct answers: "..correct_answers..")
end
```

Next, let's break it down and analyze it line by line. On the second line, the variable name is given the value io.read(). The io library has many functions that handle all sorts of input and output, but I'm using it only for keyboard input.

On the next line is the variable questions. The questions variable's value is a table that I have used like a dictionary to store both the questions and the answers. The questions are in brackets to tell Lua that they are the table's key.

Skipping the third line, there is a for loop whose function here is to use pairs() to find the key and the values of each item in the table. It then needs to place the value of the key and the value of the key's value into the variables key and value.

After printing the key (which contains the question), Lua places the user's answer through io.read() into the answer variable and checks to see whether it equals the proper answer. If the answer is correct, it adds 1 to the value of the correct_answers variable and repeats the process until there are no more items in the table to go through.

Next, Lua checks to see whether users got any of the questions correct and then prints a message telling users to learn more about UNIX (and its variants) hacker history or congratulates users on how many questions they answered correctly. Notice the concatenation operators on the 16th line.

In Python, the easiest way to do the above game would be like this:

```
name = raw_input("What's your name?\n")
questions = {"Which came first? Minix or Unix?":"Unix",
"Who created Linux?":"Linus Torvalds",
"In what year was Linux created?":"1991"}
correct_answers = 0

for key in questions:
  print key
  answer = raw_input()
  if answer == questions[key]:
    correct_answers += 1

if correct_answers == 0:
  print "You need to browse Wikipedia!"
else:
  print "\nGood job, " + name + "!"
  print "Correct answers: ", correct_answers
```

You may notice that it's easier to get keyboard input in Python than in Lua, but dictionaries are easier to identify and look much prettier in Lua. The for loop is a little more complex in Python than it is in Lua, because Python needs to know the key to be able to get the key's value. In Lua, the pairs() function breaks apart the key and its value from the dictionary table, making it much cleaner and easier to get data from tables than in Python. As for lines of code, not counting the many "ends", Lua wins hands down with 13 lines of code versus 17 in Python. Even though Lua programmers would be typing more, their code is much easier to sift through, especially when it's thousands of lines long, because of Lua's use of end rather than colons (as in Python).

Now, how about a GUI? Is programming a GUI in Lua the same, easier or more difficult than in Python? Before you try to answer that question, determine which program in the two languages will be easier to maintain, read and understand without comments. Here, I use the WxGTK library for a GUI. The only hitch with wxLua is that it is an entirely separate program. A wxLua application will not run with the regular Lua interpreter, so you must run it with the `wxlua` program.

Here's a GTK GUI program made with wxLua:

```
frame = wx.wxFrame(wx.wxNull, wx.wxID_ANY,
  "wxLua App", wx.wxDefaultPosition,
  wx.wxSize(250, 50),
  wx.wxDEFAULT_FRAME_STYLE)

frame:Show(true)
```

Now, here's the code for a program that does the same job in GTK, but with wxPython:

```
from wxPython.wx import *
class Main(wxApp):
  def OnInit(wxApp):
    frame = wxFrame(NULL, -1, "wxPython App")
    frame.Show(true)
    return true

Main().MainLoop()
```

This time, Lua's lack of necessary formatting and full-fledged OOP makes an easy job easier. Rather than create a class and function, wxLua incorporates everything needed for this GUI application in a single system function, whereas Python and wxPython require a Class as well as a function. Also note that Lua imports system libraries automatically. This wxLua application exercises some of Lua's OOP features that I discussed previously. The application creates the frame, sets the frame's name and the frame values, and then it calls the Show() function from within the wxFrame method using the colon. You also can call the frame with the period syntax rather than the colon:

```
frame.Show(frame, true)
```

Embedding and Extending

Although taking a look at embedding and extending Lua is outside the scope of this article, I touch on a few concepts here. First, the Lua API is very straightforward. Its design eliminates the need for manual reference when embedded in C code (unlike Python's API). Like the language, Lua's C API (for embedding) is fairly minimalistic. If you need advanced functionality, you can use a secondary library that is primarily made up of preprocessor macros.

Second, C and C++ are not the only languages in which Lua can be embedded. Tao.Lua provides straight .NET and Mono bindings to Lua, and LuaJava allows scripts written in Lua to manipulate Java components. LuaJava allows Java components to be accessed from Lua with the same syntax that Lua uses for accessing its native objects. It also allows Java to use a Lua interface so that any interface can be implemented in Lua and passed as a parameter to any method. The method's result (when called in the Java program) is called in Lua, and the result is sent back to Java.

Conclusion

Lua is a flexible, powerful, compact language that can be used and extended in myriad situations. Its focus on simplicity makes for easy debugging and has attracted many users. Its simple, powerful syntax provides flexibility because of Lua's metamechanisms. The small, fast interpreter uses less resources than Python, and its syntax allows for easier code readability. Its simple C API makes embedding a breeze. Whether you are doing data processing, GUIs or game programming, you will find a use for Lua.■

Joseph Quigley has been a Linux user for more than two years. He enjoys fiddling with different Linux distros and exploring new programming languages.

Resources

Programming in Lua: **www.lua.org/pil**

lua-users: **www.lua-users.org**

wxPython: **www.wxpython.org**

xwPyWiki: **wiki.wxpython.org**

wxLua: **wxlua.sourceforge.net**

LuaJava: **www.keplerproject.org/luajava**

The Tao Framework: **www.taoframework.com**

Lua Versus Python: **www.lua-users.org/wiki/LuaVersusPython**

Is GPL Java too little, too late?

Java is on its way to become mostly GPL but is it too late?

Nick Petreley, Editor in Chief

Sorry to start with the spoiler, but the answer is, "No, it is not too late." I would certainly have preferred that Sun GPL Java before Microsoft .NET was released. I think .NET would have been a total non-starter in that case. But allow me to present the evidence that Java is already kicking .NET keister on Linux.

Look at how pervasive Java has become even without the benefit of the GPL. SourceForge is one of the most if not the most popular repository of software projects for Linux (software is available for other operating systems, including Windows, but SourceForge is primarily a Linux repository). Java has enjoyed a highly prominent spot on SourceForge for a long time, well before Sun announced that it would GPL most of Java. At the time of this writing, SourceForge lists 5,421 projects written in Java. The number of Java projects outnumbers even C++ projects, at 4,582. Only 284 projects are listed for C# and only 34 projects for BASIC. I don't think it will surprise anyone that C outnumbers all others with 8,558 projects.

Here is what .NET brings to the table that Java lacks. The .NET API has richer functionality because it is not written to be a Write-Once-Run-Anywhere (WORA) platform. Java aims to be WORA, so it is missing some pretty basic features, such as good support for USB or FireWire. That's because direct support for hardware violates the WORA principle. You have to write hardware and/or platform-specific Java extension libraries that make use of the Java Native Interface to support these things.

Is that a bad thing? Must Java be faithful to WORA to be Java? In one sense, the answer is yes. The core JVM should be faithful to WORA. But that doesn't mean you can't extend Java to be platform-specific. Indeed, I hope compiled Java, in addition to the WORA JVM, gets even better as the community gets more involved in the future of Java. I am confident that only good will come of the community efforts.

I expect hardware-specific extensions to projects will flourish once Sun finishes its GPL-ization of Java. Java can remain a WORA platform for those who want to use it that way, *and* the community can provide the tools you need to use Java as a magnificent platform-specific language as well. So what if the add-ons violate WORA? It's a language, not a religion. In contrast, the Mono team is dead set on providing the non-WORA functions by playing catchup with the Windows API portion of .NET. All I can do is tip my hat and say, "good luck".

The current problem with Java on Linux that is now going away is that it isn't usually a no-brainer to install Java on your favorite distribution. That's changing quickly, but it's still hit and miss. Worse, some of the best Java applications aren't available with a simple apt-get.

The `apt-cache search jedit` command turns up nothing, even on the future release of Ubuntu, even though jEdit is a spectacular Java-based editor. A search for the BitTorrent client Azureus brings up the GCJ-compiled version. There's nothing wrong with that, but I'd still prefer to run Azureus as a Java application. I have to download and install Azureus manually if I want the non-GCJ version. There aren't all that many professional-quality standalone client applications in Java, but the ones I love and use are rarely available from standard application repositories. I expect, or at least hope, that will change as the GPL-ization of Java progresses and a Java virtual machine is installed by default on all distributions of Linux.

But here's where Java stands to explode in usage. Currently, economy Web hosting supports MySQL and PHP by default, with perhaps Perl and maybe PostgreSQL. Java usually costs extra, most likely because it's not free, and the commercial hosts who provide these services didn't get Java installed by default. I'm betting that within one year, you'll see all those $5 or more per-month hosting services provide Java and JSP by default. I've even seen a few that already have taken this step. Could I be wrong? It's certainly possible. I was wrong once before—October 1979, I think, but I could be wrong about that. Regardless, I stand by my prediction that Java will explode on economy hosting services.

The bottom line is that if you haven't already taken the plunge, do so. Java isn't as easy to pick up as, say, PHP. But, it's enough like C/C++ that the learning curve won't be overly steep if you come from a traditional C/C++ background. I've found that the key to learning Java is to focus on how to use the error-handling features properly, which is generally where I tripped up when I first tried my hand at programming in Java.■

Nicholas Petreley is Editor in Chief of *Linux Journal* and a former programmer, teacher, analyst and consultant who has been working with and writing about Linux for more than ten years.

FANATICAL SUPPORT™

Russ Barnard, President, FlapDaddy Productions

"Fanatical Support™ saved me from my own mistake."

"Not long ago, I reformatted one of our servers. Not until I was driving home did I learn that I brought our entire site down in the process. I called my guy at Rackspace and he said, 'We're already on it.' By the time I pulled in the driveway, my site was back up. Now that's Fanatical Support."

Keeping little mistakes from causing big problems is one definition of Fanatical Support. What will yours be?

Watch Russ's story at **www.rackspace.com/fanatical**
1-888-571-8976

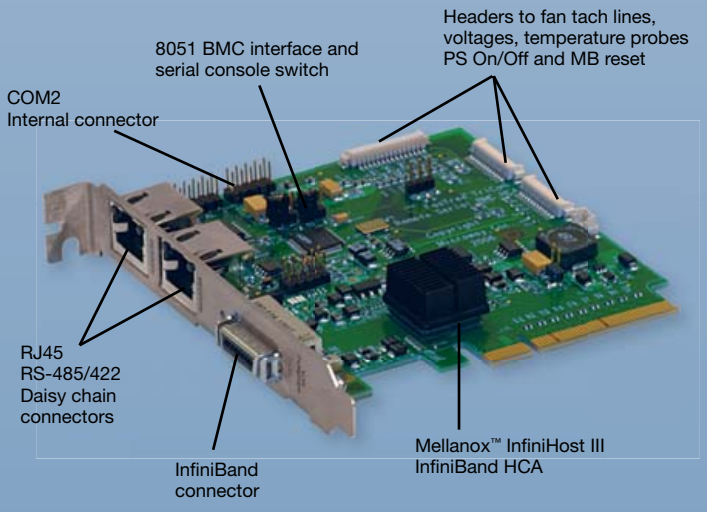
rackspace® MANAGED HOSTING

Affordable InfiniBand Solutions

4 Great Reasons to Call Microway NOW!

1 TriCom™

- DDR/SDR InfiniBand HCA
- "Switchless" serial console
- NodeWatch web enabled remote monitor and control



2 FasTree™

- DDR InfiniBand switches
- Low latency, modular design
- 24, 36 and 48 port building blocks

3 ServaStor™

- Extensible IB based storage building blocks
- Redundant and scalable
- Parallel file systems
- Open source software
- On-line capacity expansion
- RAID 0,1,1E, 3, 5, 6, 10, 50

4 InfiniScope™

- Monitors ports on HCA's and switches
- Provides real time BW diagnostics
- Finds switch and cable faults
- Lane 15 interface
- Logs all IB errors

Upgrade your current cluster, or let us design your next one using Microway InfiniBand Solutions.

*To speak to an HPC expert call **508 746-7341** and ask for technical sales or email **sales@microway.com***
www.microway.com

Microway®
Technology you can count on℠