

PHP Creator: Rasmus Lerdorf



Rasmus Lerdorf (born November 22, 1968 in Qeqertarsuaq, Greenland) is a Danish-Greenlandic programmer and is most notable as the creator of the PHP programming language. He authored the first two versions. Lerdorf also participated in the development of later versions of PHP led by a group of developers including Andi Gutmans and Zeev Suraski, who later founded Zend Technologies. In 1993 he graduated from the University of Waterloo with a Bachelor of Applied Science in Systems Design Engineering. Since September 2002, he has been employed by Yahoo! Inc. as an Infrastructure Architecture Engineer.

During his keynote presentation at OSCMS 2007 Conference, Lerdorf presented a security vulnerability in each of the projects represented at OSCMS this year.



Rasmus Lerdorf is talking about security with [Joomla! Developers](#) at [OSCMS 2007](#) Conference.

External links

- [Personal home page](#)
- [Do You PHP?](#) An introductory article by Rasmus Lerdorf about the past, the present and the future of PHP.
- [Oreilly.com:Online Catalog](#)

Notable Interviews

- [On O'Reilly](#)
- [On sitepoint.com](#)- Where Rasmus answers some questions put together by the [SitePoint](#) community.
- [Audio Conversation](#) from itconversations.com
- [Interview](#) on the [FLOSS Weekly](#) podcast with [Leo Laporte](#) and [Chris DiBona](#)

Rasmus Lerdorf - Resume

contact: rasmusATlerdorfDOTcom (no recruiters please)

Objective

The web is changing. It is more dynamic and more programmable than ever before. This new programmable web needs tools and scalable systems that can talk to each other in a way that is both useful and approachable. The learning curve has to be shallow and the results immediate. This is what I do.

Notable

- Original author of the [PHP](#) scripting language
- [ActiveState](#) Advisory board member (disbanded after sale to [Sophos](#))
- [GreatBridge](#) Advisory board member (Company went under)
- Apache Software Foundation board of directors in 2000
- Apache HTTP Server Core Development Team member
- O'Reilly author (*Programming PHP*, *PHP Pocket Reference*, parts of *WebMaster in a Nutshell*)
- Frequent speaker on PHP, and Open Source at conferences, universities and user groups around the world
- Named to [MIT/TechnologyReview's 2003 list of Top 100 Young Innovators](#)

Experience

Distinguished Engineer Sept. 2002 - Present Yahoo! Inc.
Sunnyvale, CA

- Infrastructure engineering and architecture
- PHP, Apache, Security, Web Services, Ajax, Web 2.0, and I18n
- Open Source compliance work
- Training and evangelism work at International Yahoo! offices
- Heavy yearly conference schedule (20+)

Time Off June 2001 - Sept. 2002 Dot.com Bomb/baby Vacation
San Francisco, CA

- Time off to play with the new baby
- finish writing the O'Reilly Programming PHP book and the new Pocket Ref.
- and attend countless conferences around the world

Research Jan 2000 - May 2001 Linuxcare Inc. San Francisco,

CA

- Large-scale app architecture and implementation
- Technical due-diligence work during acquisitions
- Speaker at Open Source Linux Expo (Sydney, Australia), Bang!linux 2000,2001 (Bangalore, India), ThunderLizard Web Design 2000 (Atlanta), O'Reilly 2000 (Monterey), Borland Conference (San Diego), Linuxworld (San Jose, New York), Linux Revolution 2000 (Seoul, Korea), OSDEM (Brussels), Usenix/European (Stockholm).

Senior Software Engineer 1999 - 2000 IBM Corporation RTP, NC

- Apache 2.0 Architecture involvement
- DAV (Distributed Authoring and Versioning) work
- IBM DB2 work
- Various Internal IBM projects
- Speaker at ApacheCon (San Francisco), IETF(WWW8 in Toronto), O'Reilly99 (Monterey), and TheBazaar (New York).

IT Consultant 1996 - 1998 Bell Global Solutions / Bell Emergis
Toronto, Canada

- Participated in the initial screening of opportunities relevant to Internet business solutions.
- Defined the conceptual and technical elements of Internet related business products and services.
- Provided technical architecture and direction.
- Built large-scale Virtual Email service.
- Built Advanced Server Farm service.

Technologies: Netscape Mail Server, Collabra, Commerce Server, Enterprise Server, Directory Server, LDAP SDK, FrontPage97 Server Extensions, Irix, HTTPD, CGI, NSAPI, ISDN, Cisco switches and routers, NetApp Filers.

IT Consultant 1995 - 1996 University of Toronto Toronto,
Canada

- Responsible for designing the university's new dial-up service (UTORDial)
- Wrote terminal server control software (Cisco TACACS+)
- Wrote SQL/HTML accounting system

Technologies: Cisco TACACS+ protocol, Solaris 2.5, C, Perl5, HTTPD, CGI,

HTML, Cisco ASM, AS5100 terminal servers, Expect, sh, CMU SNMP, RDBMS, Oracle7, Oracle7 Forms, mSQL, TCP/IP Internals, PPP Internals, Corba, ORB, ILU.

IT Consultant 1994 - 1999 Lerdorf Consultants Inc. Toronto, Canada

- Developed PHP/FI package which was distributed under the GNU General Public License (GPL).
- Co-authored full PHP/FI rewrite called PHP3. See <http://www.php.net/>. This is quickly becoming one of the web's most popular scripting languages.
- Member of Apache Development Group ([Apache](#) is the most popular Web server in the world)
- Speaker on PHP and Apache at a number of conferences around the world.
- Wrote the mod_info module for the Apache web server.
- Wrote a CGI redirection module for the Netscape family of web servers using NSAPI.
- Wrote CGI redirection patch for the NCSA web server
- Assisted in the design and development of countless home pages and commercial web sites.

Technologies: HTTPD, CGI, FastCGI, JavaScript, Java, HTML, C, Perl, YACC (Bison), lex (flex), Netscape Server API (NSAPI), Apache API, ISAPI, NCSA HTTPD internals, Linux, Solaris, HP-UX, OSF/1, Windows NT, Windows 95, ODBC 2.0, ODBC 3.0, Watcom SQL, RDBMS, Empress, mSQL, PostgreSQL, MySQL, Solid, Oracle, Sybase, Adabas-D, IMAP4, POP3, LDAP, ACAP, XML.

IT Consultant 1994 - 1995 Online Construction Toronto, Canada

- Developed graphical user interface for a Toronto Internet Service Provider
- Responsible for all technical issues involved in getting an ISP on-line

Technologies: C, Tcl, Perl, Solaris 2.4, Linux, Xylogics terminal servers.

Product Manager 1993 - 1994 Nutec Informática Mountain View, California

- Managed small UNIX/DOS software development team
- Developed a unique client/server GUI environment called Nutec Desktop
- Managed a tiered distribution network
- Attended all major UNIX/DOS/Networking tradeshow and was an

- invited speaker at SCO Forum in 1993
- Coordinated distributor training sessions
- Technical account manager for large accounts throughout USA and Canada

UNIX Programmer 1991 - 1993 Nutec Informática Porto Alegre, Brazil

- Designed and developed an object-oriented user interface builder
- Served as North American liaison for this South American company

Technologies: All Intel-based Unices, SunOS, Solaris, HP-UX, OSF/1, MS-Dos, C, Watcom C, Borland C, Microsoft C, AlphaWindow Standard, No-Loss Data Compression algorithms - LZ77 and LZ78 variants, XVT, Motif, GUI design.

Communications Engineer 1991 NovAtel Communications Calgary, Canada

- Part of "Think Tank" group called the Advanced Product Concept Division
- Designed a cellular data communications system
- 2 Canadian patents were registered in my name under the trademark Dataseeker/CDO (Cellular Data Overlay)

Technologies: DSP - TMS320 family, C51, 68HC11 hardware design, TCP/IP, SNA (IBM certified SNA technician), C, SCO Unix, Xenix.

Chief Support Engineer 1990 Digital Media Networks Toronto, Canada

- Maintained and performed troubleshooting on live news satellite transmission system
- Designed and implemented a forward error correction protocol to ensure data transmission integrity

Technologies: SunOS 4.1.3, No-Loss Data Compression algorithms.

Education

1988-1993 [University of Waterloo](#) Waterloo, Canada

- Honours Ba.Sc. in [Systems Design Engineering](#) - CO-OP Program
- Received a University of Waterloo Entrance Scholarship
- Workterms with BNR, NT, DMN, NovAtel and Nutec.

Publications

- R.Lerdorf, "[Dynamic Web Pages with PHP3](#)" WebTechniques Feb. 1998.
- R.Lerdorf, "[LDAP at Home](#)" WebTechniques May 1999
- R.Lerdorf, "[Installing LDAP](#)" WebTechniques May 1999
- R.Lerdorf, "[Configuring Email Clients for LDAP](#)" WebTechniques May 1999
- S. Spainhour, R.Eckstein, "[WebMaster in a Nutshell Second Edition](#)" June 1999 (wrote a chapter) ISBN 1-56592-325-1
- R.Lerdorf, "[PHP Pocket Reference](#)" January 2000 ISBN 1-56592-769-9
- R.Lerdorf [interviewed by L.Houston](#) for oreilly.com, Feb. 2001
- R.Lerdorf and others [interviewed by HotWired](#) (audio) Feb. 2001
- R.Lerdorf [interviewed by Nexen](#) (in French) April 2001
- R.Lerdorf, "[Scripting the Web with PHP](#)" Linux Magazine, July 2001
- R.Lerdorf [interviewed by S.Machlis](#) for ComputerWorld, Feb. 2002
- R.Lerdorf, K. Tatroe, "[Programming PHP](#)" March 2002 ISBN 1-56592-610-2
- R.Lerdorf, [interviewed for SitePoint by K.Yank](#), May 2002
- R.Lerdorf, "[PHP Pocket Reference Second Edition](#)" Nov. 2002 ISBN 0-59600-402-8
- R.Lerdorf, [interviewed for eprairie by J.Schneider](#), Apr 2003
- R.Lerdorf, interviewed in [Concepts of Programming Languages](#) textbook, Aug. 2003 ISBN 0-321-19362-8
- D.Kirkpatrick, Fortune article [How the Open-Source World Plans to Smack Down Microsoft, and Oracle, and ...](#), Feb. 2004
- P.Lavin, The Globe and Mail article [Popular PHP is hot, and it's Canadian](#), Sep. 2005

Interests

Traveling, Internet, languages, soccer, and developing Open Source software.

Q&A: PHP creator Rasmus Lerdorf

By - Sharon Machlis

February 4, 2002 (Computerworld) -- *Rasmus Lerdorf was the original creator of PHP, which has evolved from his personal home page project into an open-source scripting language used worldwide. Lerdorf was interviewed recently via e-mail by Computerworld online managing editor Sharon Machlis.*

Q: What would you like corporate IT managers to know about PHP?

A: PHP is not a major investment if they choose to use it. Obviously, being open source, there are no license fees to pay. But more importantly, the learning curve is extremely shallow and it draws on the skills the existing IT staff is likely to already have. PHP is not a new and revolutionary language. It borrows much of its syntax from languages such as C, Perl and Java.

Q: What are the major advantages of PHP? What is it best at?

A: It is a very focused language. It focuses on the Web problem. If you read PHP's excellent online documentation, you will see everything is geared towards solving Web-related problems. It was written by Web developers for Web developers.

PHP is perfectly suited for quickly creating a Web front end to just about any back-end system you can imagine. Typically, the back end would be a database, but it can also be [a Lightweight Directory Access Protocol] directory server, or [Simple Network Management Protocol]-manageable devices, just to name a few. PHP is also good at generating non-HTML dynamic content for the Web such as images, flash or [Portable Document Format] documents. Being able to pull customer information from a database and dynamically generating a professional-looking PDF invoice, for example, is something that is very easy to do in PHP.



Rasmus Lerdorf, inventor of the PHP scripting language

Q: What do you see as its drawbacks? What Web tasks do you think it's less well-suited for?

A: There really aren't any drawbacks when it comes to Web tasks, but I guess that depends a little bit on where you draw the line between Web tasks and back-end tasks associated with the Web. PHP's strength is in the front-end Web interface. As you move more towards the back end, PHP usefulness starts to decrease. For example, you would never write a full back-end database engine in PHP. You also wouldn't write any sort of middle-layer system for a three-tier architecture in PHP.

As the top presentation layer in a three-tier architecture, PHP does just fine.

Q: What's interesting in the works for near-term PHP future developments?

A: Deep in the guts of PHP, the object-oriented program [OOP] is going to be improved a bit. PHP has traditionally been a procedural language, and OOP features have crept in over the years to the point where PHP can be used as a decent OOP language. But there are some things that can be improved with respect to this OOP support.

Namespaces is another feature that is coming.

At the higher level, I think you can expect PEAR [PHP Extension and Application Repository] to start becoming much more useful. PEAR is a collection of userspace code for solving many Web-related problems that didn't belong in the core of the language itself.

You will also see some nice ways to build [Simple Object Access Protocol/Web Services Description Language] services with PHP. You can do this today, but I would like to see this become more transparent and am working towards that.

Q: Do you have any thoughts as to why someone should choose PHP over Microsoft's Active Service Pages (ASP)?

A: This is a difficult question to answer without getting into philosophical issues. I will try to avoid those and just look at it from a practical perspective:

- Windows 2000 Server: \$3,999
- Internet Security and Acceleration Server: \$5,999 per CPU
- SQL Server 2000 Enterprise Edition: \$19,999 per CPU
- MSDN Universal Subscription (w/ DevStudio): \$2,799 per developer

List prices from Microsoft.com

This can add up quickly. And you probably need to add some Internet connector licenses, maybe an Exchange server license if you are doing e-mail handling and perhaps the content management server, which is a whopping \$42,999 per CPU.

Many will of course argue this point, but I can build you a similar setup using Linux, Apache+SSL, PHP, PostgreSQL and Squid. And there are a number of good open-source content management systems I can throw in as well. They don't all come in shrink-wrapped boxes with slick documentation, but the one thing they certainly do is work.

Set up a nice load-balanced architecture using Squid as a reverse proxy on the front end to a couple of Web servers on the back end with a separate database server. Toss in some database replication for load balancing and robustness. So perhaps five machines with two CPUs in each. The software alone could cost you over \$300,000 for that, compared to \$0. In this economy, this is significant.

Then there is the fact that managing an IIS [Microsoft Internet Information Server] server has been a nightmare this year. One security issue after another.

This doesn't mean that the alternatives don't have security issues every now and then, but at least you are not paying someone \$300,000 on top of having to keep up with frequent security patches. For that kind of money, you should expect robust enterprise-strength software, which just works all the time. I don't mean to offend the Microsoft crowd out there, but that just hasn't been the case up to now.

Putting the whole Microsoft proprietary software vs. open source-debate aside, why should you pick PHP over ASP? In the end, they basically do the same things. Although some feel development environment is nicer for ASP than the various options for PHP, PHP has a lot of built-in functionality that you need to drop to COM [Component Object Model] to match in ASP, so from a development perspective it is probably a tossup.

If you are a devoted Microsoft shop and have developers who already know VBScript and the various other Microsoft technologies, you should probably choose ASP. If you are starting out fresh, you owe it to yourself and to your bean counters to take a long hard look at PHP and all the other great open-source alternative tools out there.

Q: I read an interesting debate when someone asked whether they should use Perl or PHP to develop interactive/dynamic Web content. Would you have any thoughts about the strengths of Perl vs. PHP?

A: Perl is a general-purpose scripting language, whereas, as I said, PHP is a scripting language targeted at the Web problem. As such, there are a number of things built into PHP for accomplishing common Web tasks, where in Perl you need to either roll your own or go trolling through [Comprehensive Perl Archive Network] to find someone else's implementation.

I personally use a combination of PHP and Perl for many of my projects: Perl mostly for back-end tasks, while I have PHP doing all the front-end work. Of course, you could do the front end in Perl, but it would be a bit more work.

Q: Can you estimate how many sites are using PHP?

A: Php.net/usage.php has some numbers. Basically, over 20% of the domains that answer on port 80 indicate that they have PHP installed. Whether or not they actually use PHP is difficult to determine.

Obviously, most of the open-source related sites out there use PHP. I know it is also used to some extent at MCI/Worldcom, Honda, Lycos and Amazon. Basically just count when you browse around. One in every five clicks is likely to be a PHP-powered site.

Q: I know PHP was an outgrowth of some code you wrote for your own home page. What were those original scripts designed to do?

A: It started out as a library of C code I had put together from countless CGI programs I had written in C. I got tired of rewriting the same code over and over again, and also wanted to separate my business logic from my HTML layout.

So I wrote a very simple tag parser that would parse through an HTML file looking for special markup tags and replace those special tags with the result of my business logic code written in C. I put together a number of general-purpose tools useful on home pages that made use of this framework.

These tools implemented a simple guest book, did hit analysis, tracked number of visitors and a couple of other things. They were intended to be examples of how to use this framework I had built that made it easy to add business logic written in C to a Web server and make calls into this business logic from within an HTML page.

Q: What, if anything, does PHP stand for?

A: The initial release back in 1995 was called the Personal Home Page Tools. Over the years, PHP has outgrown the notion of being a set of tools for home pages, so it was felt that we needed a better name. After much debate and a vote, the rather lame "PHP: Hypertext Preprocessor" name was picked. We tend to just say PHP, though, the same way very few people expand out the Perl acronym.

Q: Did you have any intention for those scripts to turn into an open-source project? If not, how did it happen?

A: No, I had no intention of creating a huge open-source project or even a scripting language. I wrote the initial tools because I felt I needed something like them and I couldn't find anything out there that approached the Web problem the way I wanted to approach it.

People ran across my home page and other sites I had worked on and asked how I had done various things. Having learned to program from looking at other people's code, it never even entered my mind that I shouldn't let people download and use mine.

Q: What does that feel like, to create something that becomes an international Web standard language, when you see your creation on thousands of Web sites all over the world?

A: That part of it has worn off long ago. What I like is when I get out in front of user groups in various countries and meet the real people face to face, where PHP has made a difference to them personally. This is especially true in some of the poorer countries of the world, where not just PHP but all open-source technologies have enabled them to participate in the global community on par with the rest of the world.

Interview - PHP's Creator, Rasmus Lerdorf

By [Kevin Yank](#)

The membership of the SitePoint community forums recently got together and produced a bunch of questions for [PHP](#)'s original creator, Rasmus Lerdorf. In reviewing his responses, I was pleased to discover that the man who originally put the PHP machine in motion maintains an unclouded vision of what the open source movement is all about.

He is quick to play down his contribution to what PHP is today, instead attributing most of PHP's success to the vast community of developers that have signed on to the project over the years. In a sense, Rasmus today is simply PHP's biggest fan.

But enough from me; let's hear what Rasmus had to say!

In the beginning...

SP: What was your first contact with the Open Source movement, and what is it about Open Source that got you hooked?

RL: Well, back in the early and mid-90's the term "Open Source" did not exist.

"Free Software" existed, of course, and I had been playing with [Linux](#) almost since the very first release in 1991. Previously I was using QNX and Xenix and then started to fiddle with Minix until Linux rescued me.

I don't think I was ever really "hooked" by a "movement". When you don't have the money to buy SCO Unix and you can download something that works and even find people who can help you get it up and running, how can you beat that? Religion never really played a part.

SP: What led you to develop PHP? And what do you think this language has to offer that others don't?

RL: The first version of PHP was a simple set of tools that I put together for my Website and for a couple of projects. One tool did some fancy hit logging to an mSQL database, another acted as a form data interpreter. I ended up with about 30 different little CGI programs written in C before I got sick of it, and combined all of them into a single C library. I then wrote a very simple parser that would pick tags out of [HTML](#) files and replace them with the output of the corresponding functions in the C library.

The simple parser slowly grew to include conditional tags, then loop tags, functions, etc. At no point did I think I was writing a scripting language. I was simply adding a little bit of functionality to the macro replacement parser. I was still writing all my real business logic in C.

In the end, what I think set PHP apart in the early days, and still does today, is that it always tries to find the shortest path to solving the Web problem. It does not try to be a general-purpose scripting language and anybody who's looking to solve a Web problem will usually find a very direct solution through PHP. Many of the alternatives that claim to solve the Web problem are just too complex. When you need something up and working by Friday so you don't have to spend all weekend leafing through 800-page manuals, PHP starts to look pretty good.

SP: Looking at the usage figures, there are now over 9 million domains using PHP. Did you have any idea that PHP was going to become this big? How does it feel to know that your product is probably the best alternative to Microsoft's solutions for the Web?

RL: First, to be clear, I did not develop the PHP we know today. Dozens, if not hundreds of people, developed PHP. I was simply the first developer.

PHP is very much a collaborative project. Think of it this way: you have a Web problem. You can either go to the store and buy an expensive shrink-wrapped product that may or may not solve most of your problem. Or you can get together with a couple of thousand people who have the exact same problem as you, and work out a solution that works for all of you.

Not only will you get a solution that addresses your exact problem, you'll also become part of a like-minded community where ideas and experiences flow freely. That beats any commercial product you can go buy at a store, and to me is the best way to develop this type of software.

So when people ask me what it feels like to have developed something that millions of people use, it doesn't really fit with how I view things. In the end, I am simply the first member of a community that has arisen around one approach to solving the Web problem.

SP: Who would you call your hero? Which people in or outside of IT have inspired you?

RL: I don't really get inspired by people in the metaphysical sense. But I definitely appreciate and respect a slick solution to a tough problem.

SP: During your years of PHP development, what do you think was the most important decision you had to make? Are there any decisions you made that you now wish you had decided differently?

It is tough to ask me to second-guess decisions that were made 6 or 7 years ago when PHP was used by a grand total of 1 person. Don't forget that I did not sit down to write a scripting language that would be used by 9 million domains: I sat down to solve a problem. Solving the problem by 5pm so you can go to a movie with your girlfriend leads to some aspects that aren't ideal 7 years later, when thousands of people have to work around that late-night hack you added.

The most important decision I made along the way was probably to give up control. To open up the project and give just about anybody who asked full access to the PHP sources. This brought in a lot of excellent talent, and people tended to feel a real sense of ownership. The PHP project

is probably one of the biggest out there when it comes to the number of people with commit access to the CVS repository where the code and documentation lives.

The Open Source movement

SP: The Open Source movement is still portrayed by many as "anarchistic" and some kind of "threat to society". Do you feel it will ever gain mainstream acceptance? If it does, how will the Open Source community deal with that?

RL: I guess there are two parts to this question.

As far as being mainstream, the product of this "movement" is most definitely mainstream. The "movement" built the Internet as we know it today. It built the TCP/IP stacks used in most of the operating systems people use (yes, even Windows). It built the most popular Web server in the world, along with the [DNS](#) and MTA systems that make the Internet tick. Heck, if you go back a bit, it built the entire industry. The first operating systems were all open source, because that was the only sane way to do things. You could not sell someone a big mainframe without providing the source to the brains of the thing. It was only later on that the concept of not providing the source code was introduced.

But I guess your real question is what I think of Microsoft's attempt to convince the world that large groups of people collaborating to solve problems somehow threatens the very fabric of the society we live in. And I don't think there are "many" people making this claim, as it is complete crap -- I would like to think that the world is a good place and not full of people who would propagate such a ridiculous idea. Let's put an end to all meetings of large groups of people while we are at it. They might be evil anarchists out to destroy the world.

In the end, mainstream acceptance is not the goal. The goal for most people who work on free software and open source projects is the technology itself. It is building a tool that solves the problem. It is not about ideology for most of us, and as such, mainstream acceptance only involves mainstream use of the technology. This has been achieved on many fronts already, with many more still to come.

SP: [PHP](#) gets very little mention in the mainstream IT press. Do you feel that PHP is being deliberately ignored outside of Open Source circles?

RL: PHP is not very exciting and there isn't actually much to it. It is a thin glue layer between your Web server and all the various things you might want your web server to talk to.

In the old tradition of [UNIX](#), we rely on small specialized add-on libraries to do all the heavy lifting with as little interference from PHP as possible. [ASP](#), [JSP](#) and Cold Fusion all have large companies with large advertising budgets behind them and the products themselves get bigger and more complex with every release so that customers will feel they got their money's worth. Who is going to spend \$10,000 for a floppy and a 2-page manual?

That floppy and the 2-page manual might actually be exactly what they need to solve their problem, and as such it might very well be worth spending \$10,000 on a small targeted solution like that. Small targeted solutions are of little interest to large software companies. The concept doesn't scale. Small targeted solutions with no advertising budget are of little interest to the trade rags.

So no, I don't think it is deliberate that PHP gets very little press. PHP is about as exciting as your toothbrush. You use it every day, it does the job, it is a simple tool, so what? Who would want to read about toothbrushes?

SP: The move away from GNU public license from PHP v3.0 to v4.0 has caused a stir among the Open Source community. Do you feel the new licensing model has now been accepted and understood as the best direction for PHP to take?

RL: PHP 3 was dual licensed actually. So we didn't actually move from the GPL to something else, we simply dropped the GPL part.

I don't really see the point of dual-licensing, and it causes a lot of confusion. By putting PHP solely under the [Apache](#)-style license it is under today, a lot of this confusion was resolved.

Dual-licensing doesn't really work as far as I am concerned -- the less restrictive of the 2 licenses is what people are going to use anyway. The various protections that the GPL offers are quite pointless when people can simply choose to use the software under the less restrictive Apache-style license. So it made sense to just go with the less restrictive of the two licenses.

If you take a look around, there are no significant scripting languages that are GPLed. And by GPLed I mean strictly GPLed in the single license sense. [Perl](#) is dual-licensed with the completely unrestrictive artistic license. Python has its own license. Ruby is dual-licensed with its own license. Tcl is under a BSD-style license. I don't see why PHP not being under the GPL would upset anybody.

Like the other scripting languages probably realized somewhere along the line, the GPL is not really needed. I would have no problems with Microsoft abandoning ASP and moving completely to PHP. They might embrace and extend it, sure, but at that point we would be in a purely technical race with them. That's a fight we can win, and in the end, having PHP everywhere would be a cool thing for the PHP community.

SP: To what would you attribute PHP's success? Do you feel that PHP has any major weaknesses (in comparison to other languages)?

RL: People like PHP because it solves their Web problem. As such, I don't see any weaknesses. It does the job it was designed to do.

Some people might argue that certain aspects of PHP are not as mature as those in other languages. The [OOP](#) support in PHP is an example. But in the end this has very little to do with solving the Web problem and more to do with aesthetics and language purism.

SP: Are you still actively involved in PHP development today?

I am still quite involved. I don't spend 20 hours a day on it like I did in the first couple of years, but I still fix bugs, argue with the other developers about features, and occasionally jump in and add the odd new bit here and there.

SP: Which Web server runs PHP the best? [Apache](#) - or something else? And which platform runs PHP the best? [Linux](#)/Intel, Solaris/SPARC, or another?

RL: This all comes down to what gets the most attention, I think. Most people use Linux/Intel with Apache. This means that bugs on that platform are discovered by the developers themselves early on, and the end-user is unlikely to hit something that the developers haven't run into already. Other mainstream UNIX platforms such as Solaris/SPARC and FreeBSD/Intel with Apache are right up there as well.

SP: PHP is usually paired with [MySQL](#). How much co-operation is there between the two teams in terms of development?

RL: We know the MySQL folks very well. The first database code in PHP was written for the MySQL predecessor called mSQL. The MySQL API was completely compatible with mSQL's when it came out, so right from the early days of MySQL, PHP had good support for it. The pairing works because PHP and MySQL tend to take a minimalistic and very direct approach to solving problems.

In terms of cooperation at the development level, there isn't that much actually. But not much is needed. PHP provides a thin layer that simply exposes the MySQL API to the PHP user. We bundle the MySQL client library with PHP, but that library is completely maintained by the MySQL team with little involvement from us -- except when they break the build, of course.

SP: Do you think PHP is becoming a replacement for [Perl](#)?

No, Perl is a general-purpose scripting language. PHP is specifically geared to the Web problem.

SP: What are your views on Magic Quotes and Register Globals?

RL: Register Globals is one of the features that brought people to PHP. The simplicity of creating Web applications when form and other variables were automatically available could not be beaten.

I was personally not in favour of turning Register Globals off by default. It adds very little to the overall security of an application. If people do not check data coming from the user then with or without Register Globals enabled that application is going to be insecure.

The only time having Register Globals off helps is when you forget to initialize a variable before you use it and someone who knows your code exploits that. By changing the error reporting level

you can have PHP find these cases for you automatically. So in the end, all I think turning Register Globals off has done is make writing PHP apps more complicated.

And it has of course also generated 10-20 questions/bug reports per day from users who are confused about this change.

Magic Quotes stems from the days when PHP was used almost exclusively for database-driven applications. These applications would take form input and stick it into a database. Even today, a large chunk of the PHP scripts out there do little more than this.

You always have to escape quotes before you can insert a string into a database. If you don't, you get an ugly SQL error and your application doesn't work. After explaining this simple fact to people for the 50th time one day I finally got fed up and had PHP do the escaping on the fly. This way the applications would work and the worst that would happen is that someone would see an extra \ on the screen when they output the data directly instead of sticking it into the database.

Often people didn't even notice this extra \ since it did not cause any fatal SQL errors and thus I wouldn't get confused emails asking me what was going on. This was a very good thing.

Even today you still see the odd site where it is obvious that the author didn't realize that data needed to be escaped before being inserted into a DB, and you see the odd extra \ here and there. Each of those is a support message we didn't have to answer.

The clueful who don't like this feature can simply turn it off and handle all escaping themselves. And the clueful who write portable apps can simply check the setting using `get_magic_quotes_gpc()` and add an `addslashes()` call when appropriate.

SP: Do you think there is a successful balance between the commercial and open source elements of the PHP community?

RL: I think it works out ok. The various commercial entities pay individuals to work on parts of PHP -- and that benefits everybody.

SP: What's been the most surprising or innovative use of PHP you've seen on the Internet?

RL: I keep seeing new and weird things, the latest being Wez Furlong's [ActiveScript SAPI module](#), which lets you do [client-side](#) PHP like this:

```
<html>
...
<script language="ActivePHP">
  function clickit() {
    $GLOBALS["window"]->open("http://www.php.net");
  }
</script>
...
```

```

</html>
```

Alan Knowles' [PHPMole IDE for PHP](#) written in PHP-GTK is quite impressive as well. There are plenty of other cool PHP things out there, but these are probably the furthest from what I started out doing.

The Future of [PHP](#)

SP: Are there any plans for server-side, stateful variables in PHP? It would be useful to place instantiated objects in shared memory so that users didn't have to incur large overhead due to class instantiation.

RL: The [Alternative PHP Cache \(APC\)](#) project can already stick instantiated classes in shared memory, so that one has been solved by APC and others.

Personally I think if you really have performance issues, you should either simplify your code a bit or have a look at writing the critical parts in C. Extending PHP at the C level is actually much easier than most people think.

SP: Current 'session' variables use disk space (e.g. /tmp) which is no good for high-traffic sites. Are there plans to remedy this?

RL: Right from day one of the session support in PHP, we provided a shared memory backend session handler. Just set your handler to `mm` instead of `files` in `php.ini`. However, for high-traffic sites this is not the solution. The real solution is to load-balance the site across multiple servers.

Having session data in memory on a single machine doesn't solve anything. For this, you write yourself a session save handler and stick your session data into a central database of some sort. See http://php.net/session_set_save_handler.

SP: What about database connection pooling? Persistent connections are not nearly good enough - are there plans to implement connection pooling in the future?

RL: A pool of connections has to be owned by a single process. Since most people use the [Apache](#) Web server, which is a multi-process pre-forking server, there is simply no way that PHP can do this connection pooling. It has to be done by a dedicated standalone process and is quite outside the scope of PHP itself. Both [SQLRelay](#) and [SRM](#) can be used to solve this problem.

If/when the common architecture for PHP is a single-process multithreaded Web server, we might consider putting this functionality into PHP itself, but until that day it really doesn't make much sense. Even Apache 2 is still going to be a multi-process server with each process being able to carry multiple threads. So we could potentially have a pool of connections in each process.

SP: Are there plans to improve [OOP](#)? Users feel there should be less overhead in instantiating classes; and the provision of encapsulation so that they can keep member variables hidden (promotes better programming). Are there any plans to this effect in the pipeline?

RL: Yes.

SP: Sybase and MS SQL Server provide support for multiple result sets returned from SQL stored procedures. PHP does not support this! When can users expect it?

RL: When someone contributes the code to do it.

SP: Database queries are currently buffered in memory before being available to the client. Can PHP programmers expect this behaviour to change so that queries are available immediately as rows are being sent from the server, so they do not have to wait?

RL: When the various database API's support this, sure they can. We already support this with [MySQL](#) through the `mysql_unbuffered_query()` call and have for quite a while.

SP: ZendEngine2 has plans for a number of exciting new features, such as Exception handling and mature OOP support. Can you give us a rough estimate as to when PHP users can expect a release - at least in terms of months or years?

RL: Nope. It will be released when it is ready.

SP: Do you see a future in PHP-GTK, with popular desktop applications being written in PHP?

RL: I see PHP-GTK mainly being used in cases where you need to provide both a Web interface and a [GUI](#) interface to the same application. Being able to use the same backend code for both is a big win.

SP: Is development of PHP and Apache now running in parallel? And is it likely that the two projects will merge in some way, in the future?

RL: PHP and Apache have always been quite closely linked since they are both pieces of the puzzle that solves the Web problem. As such there are a number of developers who work on both projects. But no, the projects will definitely not merge. That wouldn't make much sense.

SP: Do you think major corporations will use PHP in their environments instead of [J2EE](#) and [.NET](#) in the future?

RL: Some do, so yes.

SP: What would you say to young developers who are considering starting an Open Source project themselves?

RL: I am not sure it is possible in that sense. Sort of like sitting around watching your phone, trying to will it to ring. It always rings when you are in the shower or at some other inconvenient time.

I don't think you really sit down and decide to start an open source project. This "movement" is a bit of a myth that people like to glamorize and assign all sorts of unrealistic characteristics to. Nobody is going to join your project if all you have is a cool idea. Everyone has cool ideas.

People will pool around your efforts if you build something that is useful enough that they find it easier to take your code and extend it a little bit to solve their problem. If your stuff is half-baked then people are likely to dismiss what you have done, and just solve their problem themselves or by using something else out there.

So to start an open source project, first assume that you are completely on your own and solve some problem that has been bugging you for a while. That means months and months of work to get something that actually works and solves the problem. At that point you can start considering whether you might want to give up control and let others join your effort.

The key phrase here is "give up control". Starting an open source project does not mean you suddenly get a staff of programmers you can boss around. In fact, to get it off the ground you will have to be very receptive to suggestions from early adopters and do everything you can to make your tool more useful to a broader audience. And then you give it all away and let people do just about whatever they want with your code. Now you have started an open source project.

SP: What other open source initiatives have you got planned? Given infinite time (and 100 extra pairs of hands) what would you love do?

RL: Well, I didn't plan [PHP](#). I think in terms of solving problems, not in terms of software projects. I actually hate programming, but I love solving problems. So what problems would I like to solve if I had lots of time and resources?

Having recently become a father I had grand plans to build a smart-crib. Live video, audio and various other gadgets that could be viewed and controlled remotely to let remote friends and family interact with the kid. Of course, once the baby actually arrived finding enough time to just sit and read a book is nearly impossible, never mind building a smart-crib!

Another really cool project would be the ultimate distribution building system. One where I could specify that I had a device with an 80486 processor, a certain type of NIC, and whatever other hardware specifics along with the type of media and ram, and this thing would crank out a small [Linux](#) distribution tailored exactly for my device.

Then, to extend that, be able to say that I want it to act as a firewall, mp3 server, browsing station or whatever. Basically get to the point where you can take almost any hardware device and stick Linux on it and have it be useful. I have a lot of devices around the house that I know I can put to better use simply by putting more powerful software on them, and I'd like to find a way to do this without having to spend 6 months on each one.

The SitePoint community and I would like to take this opportunity to thank Rasmus for his time and detailed answers to the inquiring minds in our community. He may no longer be the controlling force behind PHP, but he definitely shines as a member of the team that is working together all over the world to take PHP to the next level.

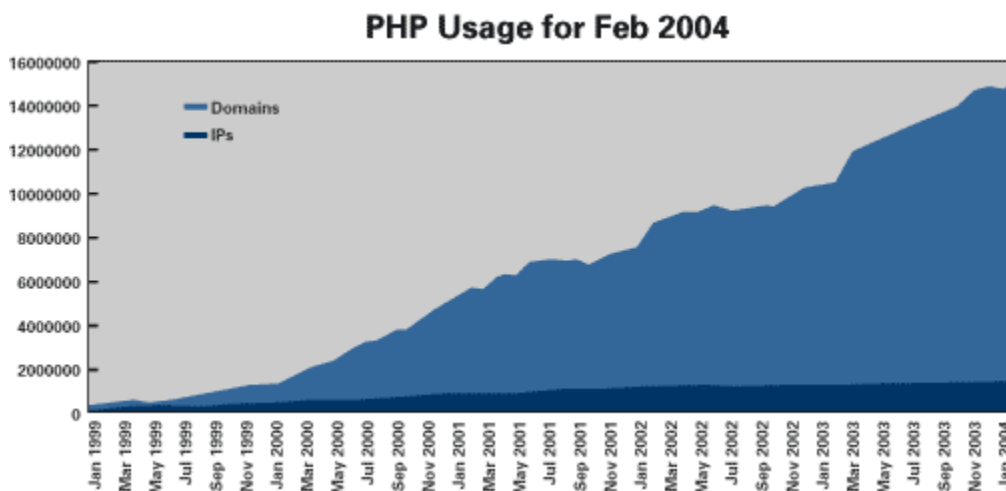
The Hitchhiker's Guide to PHP

Do You PHP?

By Rasmus Lerdorf

PHP's creator offers his thoughts on the PHP phenomenon, what has shaped and motivated the language, and where the PHP movement is heading

PHP is everywhere. In its February 2004 Web server survey, Netcraft [www.netcraft.com] poked 47,173,415 domains and found that 15,205,474 had PHP installed. That is approximately 32 percent of all domains on the Web, and there is no sign of its slowing down.



[phpstats.png]

The Early Days of PHP

I started developing PHP nearly 10 years ago now. That was long before the term "Open Source" was coined and before the GPL and Free Software was well known. And as with many open source projects that have gone on to become popular, the motivation was never philosophical or even narcissistic. It was purely a case of needing a tool to solve real-world Web-related problems. In 1994 the options were fairly limited when it came to Web development tools. I found myself writing dynamic components for Web sites in C or Perl, and the code overlap from one problem to the next was quite significant. For performance reasons, I was increasingly tending away from Perl and toward C, because the fork+exec overhead of having to run Perl as a standalone CGI was too restrictive.

The initial unreleased version of PHP was mostly a C library of common C functions I had written to be easily reusable from one open source project to the next. I had a simple state-machine-driven parser that picked tags out of HTML files and called the back-end C functions I had written. This code was initially released to the public as a package called Personal Home Page Tools, and each tool in the package was an example of how to use the system to solve a common problem on a personal home page. At some point along the way, I split out a piece and

called it FI, for Form Interpreter. The idea behind FI was that it could do all the common things you needed to do when you received the result of a form submit. Some early examples:

```
<!--getenv HTTP_USER_AGENT-->
<!--ifsubstr $exec_result Mozilla-->
Hey, you are using Netscape!<p>
<!--endif-->

<!--sql database select * from table where user='$username'-->

<!--ifless $numentries 1-->
Sorry, that record does not exist<p>
<!--endif exit-->

Welcome <!--$user-->!<p>
You have <!--$index:0--> credits left in your account.<p>

<!--include /text/footer.html-->
```

My parser for FI was terrible, which prompted me to try to write a better one. I moved away from the `<!-- cmd -->` syntax and went to `<? cmd >` instead, recombined parts of the Personal Home Page Tools with this new FI tool, and released that as a package called PHP/FI (the name was a bit of a tongue-in-cheek play on TCP/IP) in late 1995. PHP/FI grew right along with the Web over the next couple of years. In 1997 two guys in Israel, Zeev Suraski and Andi Gutmans, who were using PHP/FI asked if I would be interested in using a new parsing engine that they would write for the next version of PHP. I gathered up a few other people who had been sending patches and code for PHP/FI and we all coordinated to release PHP version 3 in mid-1998. This was probably the most crucial moment during the development of PHP. The project would have died at that point if it had remained a one-man effort and it could easily have died if the newly assembled group of strangers could figure out how to work together towards a common goal. We somehow managed to juggle our egos and other personal events and the project grew. The number of contributors has grown steadily and today we are pushing towards a release of PHP 5.0 sometime in the first half of 2004.

The Ugly Duckling of Programming Languages

Popular opinion about PHP is polarized. Language purists tend not to like the somewhat haphazard implementation of many features and some of the inconsistencies that have emerged over the years. At the same time, pragmatic problem solvers tend to love how PHP seems to almost read your mind and present itself as the perfect Web problem solving tool.

Among the things that drive the purists crazy are that names of functions are not case-sensitive but variables are; built-in functions are not consistently named; and no real structure is enforced on PHP developers, making it easy to write messy code. I can't really argue with these criticisms, but I can at least attempt to explain how and why we got to this state.

First, regarding the function-name case-sensitivity issue: This goes back to the very first version of PHP. In the early days of the Web, long before XHTML, it was very common for all HTML markup tags to be uppercase. But because these tags were not case-sensitive, people weren't very

consistent about this. I wanted people to treat the special PHP tags as being basically just like other markup tags, which meant that PHP's tags should also not be case-sensitive. As PHP became more advanced and received features such as variables, it didn't hurt to make these new features case-sensitive, because it didn't break backward compatibility with existing PHP pages. Going back and suddenly making the initial simple tags, which were in essence just function calls, case-sensitive would have broken those pages and made them unusable in newer versions of PHP. People shouldn't have functions whose names differ only in case, anyway. Still, in retrospect, it would have been a good idea to break backward compatibility early on, when relatively few people were using PHP, but at the time, nobody could have predicted the amazing growth of PHP.

As to function naming itself, I tended to steal/borrow ideas from other languages and APIs I was familiar with. This means that PHP has functions such as `strlen()` and `substr()`, which would look silly if written as `str_len()` or `sub_str()`. I added things like `stripslashes()`, which, because of the length, is often written as `StripSlashes()` to make it easier to read. At the same time, I mimicked low-level database APIs, with functions such as `mysql_connect()`—miniSQL was the first database to be supported by PHP—which used underscore naming. People familiar with these various sources were quite at home with the naming in PHP. PHP was never so much a standalone language as it was an interface between the Web server and all the various back-end tools you wanted to hook into the Web server. Consequently, when people look at PHP today as a standalone language without taking its context into account, it can appear somewhat inconsistent.

About the lack of enforced structure, all I can say is that I absolutely hate programming frameworks that lock me into a certain way of approaching a problem. That doesn't mean I don't believe in structure and frameworks, but I do believe in people having the power to come up with their own to match their environment. More on this later in the article, when I discuss possible architectures for your various PHP projects.

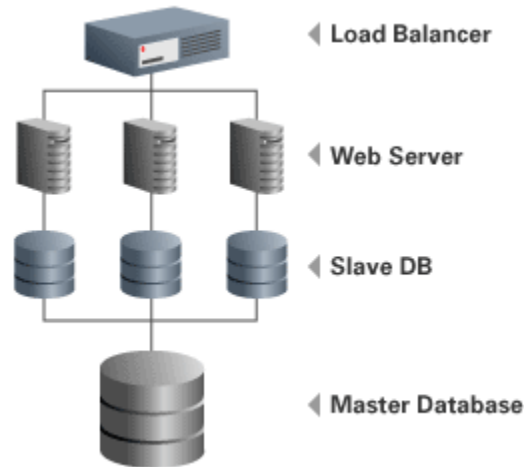
<p style="text-align: center;">PHP Resources</p> <p>PHP Web site PHP documentation PHP code repository PHP extensions</p>

What it all boils down to is that PHP was never meant to win any beauty contests. It wasn't designed to introduce any new revolutionary programming paradigms. It was designed to solve a single problem: the Web problem. That problem can get quite ugly, and sometimes you need an ugly tool to solve your ugly problem. Although a pretty tool may, in fact, be able to solve the problem as well, chances are that an ugly PHP solution can be implemented much quicker and with many fewer resources. That generally sums up PHP's stubborn function-over-form approach throughout the years.

Advice for Architects

The most popular deployment model for PHP is for it to be linked directly into the preforking multiprocess Apache 1.3.x Web server. Unlike with Java, there is no standalone process like the JVM. PHP is similar to scripting languages such as Perl and Python in that it parses and executes scripts directly.

The lack of a central control process is a feature and at the same time a source of great frustration for many. The shared-nothing architecture of PHP where each request is completely distinct and separate from any other request leads to infinite horizontal scalability in the language itself. PHP encourages you to push scalability issues to the layers that require it. If you need a shared datastore, use a database that supports replication and can scale to the levels you need. If you need to load balance requests or distribute certain requests to certain servers, use a front end load balancer that supports this. By avoiding a central controlling process, PHP avoids being the bottleneck in the system. This is the defining characteristic that separates PHP from what people commonly refer to as application servers.



[oarch1.png] A very common architecture for a high-end PHP deployment

In the diagram above, one or more load balancers distribute incoming requests across any number of Web servers. For data storage, you might deploy a read-only database replica on each Web server if your dataset is small enough to let you do that, or you might create a separate tree of database servers to handle various types of requests.

Adding Structure

One of the big strengths of PHP over many other tools aimed at solving the Web problem is that other tools tend to associate such very specific targeted problem solving with the need to control how users approach the problem structurally. PHP doesn't impose any such structure, choosing instead to focus on making each individual functionality aspect of the problem as easy as possible to use. For example, PHP provides very targeted functions for communicating with a back-end database. These are specific to each database and do not sacrifice any performance to gain uniformity or consistency with other back-end databases. There is also no set way to structure a PHP application in terms of file layout and what goes where.

The fact that PHP doesn't impose structure doesn't mean that you shouldn't build your PHP applications in an organized and structured way. Here is one approach I like to show people who ask me how I would go about structuring a large PHP application.

+-----+

HTML TEMPLATES	\$DOC_ROOT/*.php
TEMPLATE HELPERS	\$DOC_ROOT/*.inc
BUSINESS LOGIC	/usr/local/php/*.inc
C/C++ CORE CODE	/usr/local/lib/php/*.so

This four-layer approach addresses a couple of issues. First, it separates the content along the lines of responsibility in a typical project. The Web front-end developers work from the top, and the back-end engineers work from the bottom. They overlap a bit in the template helpers layer. It also separates anything that contains HTML, by putting those files into the `document_root` and anything that doesn't contain HTML outside the `document_root`.

PHP Tips and Tricks

1. When you are looking for information on a specific PHP function, go to <http://php.net/> <function_name>. For example: <http://php.net/join>. That takes you directly to the right place in the online manual on a server geographically near you.
2. Try this: `pear install apc`. The `pear` command is a useful installer that can even be used to install PHP extensions written in C. In this case, it would install the APC opcode cache extension.
3. Use an opcode cache to improve performance. See #2.
4. There is nothing wrong with mixing OOP and procedural code in PHP. Use objects when they make sense, and go procedural for the rest.
5. Extending PHP with your own custom C or C++ extensions is easier

The top template layer typically contains very little PHP—just simple function calls and the odd include. Perhaps a loop. These files are usually edited with an HTML authoring tool. The second layer, the template helpers, is where the interface between the business logic and the layout is defined. This layer might have convenience functions such as `start_table()`, `show_user_record()`, and any other reusable component that makes template authors' lives easier.

The business-logic layer does not contain any HTML at all. This is where such things as SQL queries and any other PHP user-space business logic is implemented. You might expect to see a function such as `get_user_record()` implemented in this layer. This function would take an ID, perform the appropriate SQL query, and then return an associative array with the result. A function in the layer above then takes this array and wraps some HTML around it to make it look nice.

The final C/C++ layer is where you put any custom back-end code required for a project. Many people will not have anything for this layer, but if you have a proprietary C or C++ library, you can write a PHP extension to interface to that here. Sometimes this layer is also used when a business-logic function written in user-space PHP turns out to be too slow.

Hiring and Training PHP Developers

PHP is not a new language. It doesn't introduce any new concepts. This means that training programmers who already know any C, C++, Perl, or even Java to write PHP code is quite easy. I tend to look for people with C or C++ skills when I go looking for PHP developers for a project, the thinking being that you are much better off hiring experienced programmers than you would be if you hired someone who necessarily knows a lot about PHP. If they can handle those languages, PHP will be trivial for them. Of course, if they have experience with both, so much the better.

Deploying PHP Everywhere

Use the right tool for the job. I have run across companies that have completely bought into PHP, deploying it absolutely everywhere, but it was never meant to be a general-purpose language appropriate for every problem. It is most at home as the front-end scripting language for the Web. Depending on the traffic the Web site gets, it can be used to do the bulk of the back-end

than you might think. See `README_EXT_SKEL` in the PHP source distribution.

6. The `echo <<<EOB` syntax is useful for outputting blocks of text with full `$variable` substitution, without needing to escape anything. `EOB;`

7. `PATH_INFO` is good! Use it to clean up ugly URLs.

8. Use a profiler, `pear install apd`.

9. Database abstraction is mostly a myth. There is nothing wrong with direct database calls' making use of all the tricks and cheats your chosen database has to offer, to tweak as much performance as possible out of it.

10. Keep your base technology and building blocks simple. Stay with a nonthreaded Web server, and avoid complex frameworks and abstraction layers, to give yourself a chance to trace and debug any problems that may come up. Solving the Web problem is simple; don't try to make it hard.

work as well. But at a certain point, you are going to need to write part of your code in a strongly typed, compiled language such as C or C++ for optimal performance.

Where Is PHP Going?

People are always asking me what lies ahead for PHP. That is a very difficult question to answer, because PHP is mostly a reactive open source project, in that it evolves to meet the needs of its community. In PHP5, the OO capabilities and integration with XML have been improved greatly. We have integrated an interesting tool called SQL-Lite, which provides a SQL interface directly to a file without requiring a server. It is not a replacement for a real database, obviously, but using it is certainly a much better approach than trying to write your own flat-file manipulation routines. And the fact that it gives you a SQL interface means that migration to a real database becomes easier if that is ever required.

These changes in PHP5, although significant, are evolutionary. We are not turning the world of PHP upside down with this release. Of the scripts written for PHP4, 99 percent will work unchanged in PHP5. The biggest change is that objects are handled differently in PHP5. When you new an object in PHP5, you now, by default, get a reference to that object that you can pass around without explicitly stating that you want the object passed by reference, as you had to in PHP4. If you actually want a copy of the object in PHP5, you need to "clone" it.

Longer-term, there are people exploring the use of the Parrot engine. Parrot was written as the engine behind Perl6, but it is really a language-neutral general-purpose scripting engine. It would be very interesting if the various scripting languages could all agree on a single back-end engine, which could then be used as a basis for common extensions and much better language interaction.

And still others are exploring Java connectivity through JSR 223, with some thinking that Java can be the single universal back end for scripting languages.

Despite what the future may hold for PHP, one thing will remain constant. We will continue to fight the complexity to which so many people seem to be addicted. The most complex solution is rarely the right one. Our single-minded direct approach to solving the Web problem is what has set PHP apart from the start, and while other solutions around us seem to get bigger and more complex, we are striving to simplify and streamline PHP and its approach to solving the Web problem.

Rasmus Lerdorf (rasmus@lerdorf.com) was born in Godhavn/Qeqertarsuaq on Disko Island off the coast of Greenland in 1968. He has been dabbling with UNIX-based solutions since 1985. Known for having gotten the PHP project off the ground in 1995, the mod_info Apache module and he can be blamed for the ANSI92 SQL-defying LIMIT clause in mSQL 1.x which has now, at least conceptually, crept into both MySQL and PostgreSQL.

He tends to deny being a programmer, preferring to be seen as a techy adept at solving problems. If the solution requires a bit of coding and he can't trick somebody else into writing the code, he

will (very) reluctantly give in and write it. He is currently an infrastructure engineer at Yahoo! Inc. in Sunnyvale, California.