www.phparch.com

MAY 2009
VOLUME 8 - ISSUE 5

php architect

PHP ON Microsoft® Windows

Looking Better than Ever

Introduction to Silverlight and the HTML DOM
Access a Plethura of New RIA Possibilities

Installing PHP on Windows
Set Up Your Development Environment in Minutes

Configure and Optimize PHP on Windows
Improve Your Speed and Efficiency

Getting Started with the SQL Server Driver for PHP
Harness the Power of SQL

Pre-Launch Checklist
Is Your Site Really Ready to Go Live?

ENTERPRISE PHP
The Rise of Frameworks

INSIDE

pear CORNER

php architect
www.phparch.com

CONTENTS
MAY 2009

FEATURES

COLUMNS

Download this month's code at:
http://phparch.com/code

WRITE FOR US!

If you want to bring a PHP-related topic to the attention of the professional PHP community, whether it is personal research, company software, or anything else, why not write an article for php|architect? If you would like to contribute, contact us, and one of our editors will be happy to help you hone your idea and turn it into a beautiful article for our magazine. Visit www.phparch.com/writeforus.php or contact our editorial team at write@phparch.com and get started!

Stay Connected

by Elizabeth Tucker Long

Greetings to everyone getting together at one of the many conferences this month, especially at php|tek. I wish I could be there! Conferences are a great place to not only sample new technologies and methods, but also to meet your fellow developers. Too often, programmers work in a vacuum, learning new things from books or online documentation by themselves. Don't get me wrong, this kind of self-reliance is a great skill, but I have to admit, I love to see programmers hanging out together at conferences, getting to know new people, and discussing solutions to problems they've faced. So, do me a favor, get out and meet some people this month. Not sure where to do it? I highly recommend php|tek (May 19-22, 2009 - http://tek.mtacon.com) if you can swing it. Need something local? Check out our directory of user groups in this issue. Your user group not listed? Send an e-mail to beth@phparch.com, and let me know the details so we can include it in our directory.

This month, we're focusing on using PHP on Windows. Now, in the past, the PHP + Windows relationship has gotten a bad rap, but thanks to a collaboration between the PHP Community and Microsoft, in the form of the PHP Windows Team, this relationship has really taken off, and we hope that we can bring to light some of the great new tools and features available to PHP programmers. To get you started, Hank Janssen and Pierre A. Joye walk you through installing PHP on Windows. Once you've completed installation, check out Ruslan Yakushev and Hank Janssen's article on optimizing PHP on Windows to gain some speed. David Sceppa's article introduces you to the SQL Server Driver for PHP, and Joe Stagner walks you through adding the power of Silverlight to your PHP apps. If you've got a site going live, you'll want to check out Eric David Benari's pre-launch checklist to make sure it's really ready. Is your site or project still in the planning stages? Be sure to take a look at Arne Blankerts' column on the cost of security and Helgi ormar orbjrnsson's column on quality assurance teams, both essentials items to discuss when planning a new project. Ivo Jansch interviews Matthew Weier O'Phinney to find out why frameworks are so popular (and if you can't get enough of frameworks, stay tuned next month for our frameworks superissue!), Derick Rethans walks us through part 2 of his column on PHP's garbage collection mechanism, and Marco Tabini wraps things up for us with a discussion on the loss of simplicity and creativity in web design.

May 2009
Volume 8 - Issue 5

Publisher
Arbi Arzoumani

Editor-in-Chief
Elizabeth Tucker Long

Author Liaison
Elizabeth Naramore
Cathleen MacIsaac

Technical Editors
Simon Harris
Ian Lessing
Clark Everetts

Graphics & Layout
Arbi Arzoumani

Managing Editor
Arbi Arzoumani

Authors
Eric David Benari, Arne Blankerts, Nik Garkusha, Ivo Jansch, Hank Janssen, Pierre Joye, Helgi Þormar Þorbjörnsson, Derick Rethans, David Sceppa, Joe Stagner, Marco Tabini, Ruslan Yakushev

php|architect (ISSN 1709-7169) is published twelve times a year by Marco Tabini & Associates, Inc., 28 Bombay Ave., Toronto, ON M3H1B7, Canada.

Contact Information:
General mailbox: *info@phparch.com*
Editorial: *editors@phparch.com*
Sales & advertising: *sales@phparch.com*

Printed in Canada

e-xact transactions

PAYMENT PROCESSING TECHNOLOGY

MOVING BUSINESS. REAL TIME. REAL SMART

They don't get IT!

- Banks do what banks do.
- Take your money.
- Use your money.
- Charge you to access your money.

We get IT!

- We build software.
- Our code works.
- Download via the Web.
- Bank Agnostic.
- 24 x 7 Test System

You get IT!

- **FREE**
- *http://developer.e-xact.com*
- Influence our development

E-xact Transactions (Canada)
Suite 304 - 134 Abbot Street
Vancouver, BC, V6B 2K4
Canada

Contact Information
Toll Free: 1-877-303-9228
Ph: 604.691.1670
Fax: 604.691.1678

Web: www.e-xact.com
Email: comments@exact.com

Trading Symbol: EXZ.U
Listing: (CDNX)

QA Teams
Truth or Myth?

by Helgi Þormar Þorbjörnsson

In the last couple of years, there seems to be a growing trend in the PHP world where application and library developers put out claims that their piece of code is high-quality and thoroughly tested. Indeed, in many cases, this is more than true, but how can you really guarantee quality like this?

Quality Assurance, What Is It All About?

You may be wondering, what is Quality Assurance (QA) all about? Is it something you might want to start utilizing for your project?

I don't want to go too deep into this topic since this is a mere short column but QA can be so many different things depending on who you ask. I'm going to take the plunge and attempt to give my thoughts on QA teams and hopefully enlighten you in one way or another along the way.

The QA Team

So what is the QA team? They are the keepers of quality, those who aid the developers in maintaining the balance between productivity and chaos, these are the people that strive for perfection – The QA Team *cue theme song for "The A team"* Modern day heroes.

From my perspective, the responsibilities that a QA team would take on range from monitoring the ticket system, gathering more accurate information for tickets, verifying tickets, and making sure developers are upholding the coding standards, and any other standards, the project is employing, just to name a few.

One important purpose of QA teams is to think up new ways to make it easier for developers to adhere to the standards, to make it really easy for end users to report issues and to find information they need to reduce duplicate reports or invalid ones. To create a better end user experience usually requires the QA team to bother the developers to write proper API documentation and harass the documentation team to produce a proper user manual. In many projects, the QA team will also deal with the tests – Writing user stories based on the specifications (if they are so lucky to have such a thing) so that the developers can write accurate functional and unit tests. This will also allow the QA team to do proper verifications before releases and sprint closures.

QA teams are an impartial third party, they have an outside perspective on the application. The problem with developers is that they don't make good users due to how close they are to the product. They know what can be done and what can't be done and are thus less likely to be as creative as end users when it comes to using/testing the product, QA is the bridge between the two.

QA will point out things that work but are not as clear or obvious as they could be, e.g. obscure error messages, unintuitive interface (be it user interface or API design) or finding fragile code where something is a little off and ends up blowing up in your face.

This is just the tip of the iceberg and, as you may have noticed, QA teams need to able to touch every aspect of the product, know it inside out, and play nicely with everyone inside the project. They are indeed the

most powerful weapon a project manager can have in his/her arsenal, and if used correctly, the whole team will be more productive and produce higher-quality products on time ... Well that's the general idea, at least. ;-)

They Are Around – Even When You Think They Aren't!

More often than not, projects will have a QA team, or at least QA people, without anyone involved even knowing about it!

Just think for a moment about all of those projects that you have come across that are high-quality and are well managed, yet have no official QA team nor any mention of QA anywhere.

This scenario tends to happen when you have smaller companies or smaller development teams within companies. They become their own QA team, usually without appointing someone to take care of the bug system, with no one doing proper bug triage sessions and with verifications simply happening when someone tries to fix the bug and ends up chasing down the person that filed the report just to get enough information to be able to actually fix the problem. This is all highly inefficient but still displays an effort to maintain a certain amount of quality.

QA and PEAR

A good 5 years ago, the PEAR QA efforts fell into the category: "It is around, even when you think it isn't." The project had a QA mailing list where there were some people around that did the occasional poking about and kept the project in decent shape. Quality was by no means bad, but as the project grew in size, the demand on high-quality code rose and since PEAR had some standards in place already, it became evident – A QA team would have to be formed to take on this challenge in a more official and more efficient way.

With that in mind, an RFC was written by Lukas Smith and yours truly (http://pear.php.net/pepr/pepr-proposal-show.php?id=60) on various things in regards to operating the PEAR QA team. The RFC was accepted and became the QA team's modus operandi which still holds true to this day.

Sadly, as is often the case with open source projects and indeed even companies, resources and man power are scarce, and as such, a QA team might not be in place from day one or it's just too overwhelming for the team at hand and things start piling up, which was the case for PEAR.

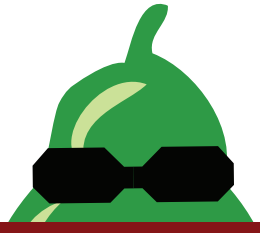
Bug reports as old as 6 or 7 years were still open and unattended, something not acceptable by a project claiming high standards, and so last year (2008), after brainstorming, we decided to attempt a bug triage bi-monthly in the spirit of the Mozilla-run bug days, where end users and developers come together on a specific day and go through old bugs or a specific portion of the system(s) decided upon in advance, verify reports, write test cases and fix code.

I ran the first couple myself with a small group of people attending. Finding myself lacking time, Daniel O'Conner stepped up to take over my role, and he has done it so well that PEAR has never before been in such a good state quality-wise!

This is just one example of how an active QA team can make a difference for a project, another excellent example would be the Test Fest the PHP team put together for 2008, which was all about involving the user groups and communities to help write new test cases and get the coverage up for PHP itself.

As a result of Test Fest, PHP received over 158 new tests, a truly great effort to combine test writing and community involvement. It went so well that it will be repeated this year (See Zoe Slattery's article in the March 2009 issue of php|architect).

The moral of the story is this: A QA team can bring a lot to your project and is an important part of the development process – Do not only think about delivering quickly, think about quality for the sake of your end users.



On day to day basis **HELGI ÞORMAR ÞORBJÖRNSSON** works for a new and upcoming startup, echolibre, that specializes in building APIs, creating top of the line web applications, security audits, and systems scaling. The rest of his time is taken up by his passion for all things PEAR. Helgi is a long term contributor with a wide range of PEAR packages behind him, and also takes care of the PEAR website. He is currently an elected member of the governing body, PEAR Group. Helgi enjoys writing articles, speaking at conferences and working on PEAR2 and Pyrus.

ENTERPRISE PHP

The Rise of Frameworks

by Ivo Jansch

Frameworks are more popular than ever and seem to have replaced the CMS as the implementation of choice for websites. In this article, Ivo interviews Matthew Weier O'Phinney, software architect for Zend Framework, on several topics, such as the reasons why frameworks are so popular.

The amount of articles in php|architect on various frameworks has increased significantly over the past year. This reflects the general popularity of frameworks. To analyze this trend, I have interviewed Matthew Weier O'Phinney, the software architect for Zend Framework, one of the frameworks that has grown in popularity tremendously. Although Matthew is naturally biased towards Zend Framework, we had a very open discussion on frameworks in general.

First I asked Matthew his opinion on what drives people to use frameworks. Matthew: "I think there's growing recognition that many PHP development 'problems' already have solutions, and that it's a waste of developer time to code new solutions for these. Frameworks and libraries, such as PEAR, Zend Framework, Solar, and others, have already poured countless hours into creating solutions, as well as into testing them thoroughly. Why code your own imperfect solution when there's a well-tested and community vetted one available?"

That is just one reason to use a framework. There are many more, so I asked Matthew to give me his top 3 reasons:

- Use a framework in order to make better use of your time (code your application, not its dependencies)
- Use a framework to leverage the knowledge of the masses (don't reinvent the wheel)
- Use a framework to build on a tested foundation (test your business logic, not how to fetch results from a database)

These are very much related, yet touch on different aspects of development. What Matthew's reasons have in common is that all 3 focus on optimizing development. You could sum this up as: A framework makes you more efficient.

Despite the popularity of frameworks, there are also people opposed to frameworks. One example is Rasmus Lerdorf (original author of PHP). What the opponents usually say is that PHP itself is already a framework. Any framework built on top introduces overhead. I asked Matthew what he thought of this. Matthew: "Absolutely there's overhead to using a framework. However, frameworks offer advantages over vanilla PHP. First, with a framework, you will often get a more consistent API than PHP offers. If you've ever had trouble remembering needle/haystack arguments for PHP functions, you'll understand the benefit of this."

"Second, frameworks are generally written for re-use. Most frameworks utilize OOP principles, which means that you can extend a class, and override or customize specific functionality. This is simply not possible when using PHP functions directly."

"Third, a (good) framework will also provide best practices: how to name classes, methods, variables, files, etc. This kind of consistency makes long-term maintenance of a code base much simpler."

Another reason I hear for not using a framework is security; if someone discovers a security hole in the framework, anybody can hack my site. Matthew: "This is true to an extent...but not entirely. Most frameworks have mechanisms for dealing with security issues, but ultimately, it's the developer's responsibility to code with security in mind. That said, I don't know of any mainstream framework that won't immediately issue a security patch release as soon as they are notified of a security hole. The thing is, PHP itself occasionally has security holes reported. I'd say that you are no more and no less secure using a framework than using PHP itself."

The next topic we discussed was off-the-shelf frameworks versus home-grown frameworks. When I ask audiences at conferences what frameworks they use, I usually get a remarkable answer. While a significant amount of them use a framework, the most popular

framework in such polls is often 'my own framework'. I asked Matthew what he thinks of these 'home grown' frameworks. "I've actually built my own a couple of times. There's a saying in PHP circles, I'm not sure who originated it, but it goes: 'Every PHP developer codes 2.5 frameworks'. My take on it is that the first is always very imperfect, and the developer learns from their mistakes. The second is much more sound, but not as comprehensive as it could be. The 0.5 is when they create a framework that extends on an existing one, and contribute back features and patches to the existing framework. I think there's a lot of merit to coding your own framework. You learn a lot about good software architecture, and can tailor it exactly to your site or organization's needs."

So apparently, a good approach is to combine the two: build your own, but on top of an existing one.

Matthew continues: "The problem with building your own framework is that you end up duplicating effort others have already done, and often don't realize what the pitfalls and edge cases are. So, this is where building on top of another one is good - building specific functionality on something common. There are times when it's good to have a custom framework: if your site needs high availability and excellent performance, in all likelihood, you'll need to do something custom. But that doesn't necessarily rule out re-using existing code from other projects."

Many developers say they write their own frameworks because they learn so much. Is that a good reason? "Yes, it's a great way to learn. The thing to be aware of, however, is that others *have* done it before. It's often just as good of a learning experience to sit down and browse through a framework's classes and see how it all works together. But for those who are more 'hands on' types, I highly recommend the 'code your own' approach. Just be willing to let go of your ego and adopt something else later. And if you're doing it on company time, just go and learn one or more frameworks instead – be aware that if you write your own, it will need to be maintained later as well."

Next, we went on discussing if the popularity of frameworks helps strengthen PHP's position in enterprise level projects. I've read articles saying 'Java has J2EE and frameworks like struts, PHP has Zend Framework'. Is Zend Framework (or any of the other php frameworks) PHP's shot at being recognized as 'enterprise ready'?

Matthew: "Frankly, I think PHP is already 'enterprise ready'. It's proven itself time and again, everywhere from corporate intranets to large scale sites. Why a language needs one or more frameworks to be "enterprise ready" is beyond me. The only topic that springs to mind is support and certification – but PHP has that through a variety of channels as well (Ibuildings, OmniTI, Zend, Blue Parabola, and countless others - and the ZCE certification as well). I think enterprise organizations like to have something to point at, however, and frameworks are currently 'it'. Zend Framework receives a lot of buzz in the enterprise circles in large part due to having the Zend brand."

Enterprises like terms such as 'quick time to market', that's something many frameworks bring. Maybe that's why they are so fond of them. Matthew continues: "I don't think it's the only shot for PHP in the enterprise. Right - frameworks bring RAD to PHP - but really, that's been possible for some time. RAD is one thing, ease of deployment and maintenance is another. PHP's success is its very own design, and I think that's something that the enterprise market is beginning to better understand and appreciate."

When RAD came up, I asked Matthew whether we're going to see RAD support for Zend Framework in IDE's (such as wizards to generate controllers).

Matthew: "Yes. By the time people read this article, we will have released ZF 1.8, which introduces Zend_Tool. Zend_Tool includes support for generating project infrastructure, as well as code generation for a number of different ZF class types, including controllers, views, helpers, etc."

Given the amount of good frameworks, I asked Matthew to temporarily set aside his bias towards Zend Framework and asked what is the important selection criteria when choosing a framework. How can one differentiate between the various frameworks out there?

Matthew: "There are a number of criteria to consider. PHP version is one. While PHP 4 support is discontinued, there are a ton of hosts still using it, and if you need that support, that narrows your choice (CakePHP and CodeIgniter, for instance, still support PHP 4;

Want to know more about frameworks?

The next issue is for you. Stay tuned in June for introductions to no less than **7 different frameworks!**

symfony and ZF do not). Another consideration is what type of project you're doing. If you're porting an existing project to a framework, some frameworks will accommodate that better than others. Many frameworks are 'opinionated', and make retrofitting to legacy code bases difficult; others provide flexibility for this very purpose. For 'green field' projects, you often want to get up and running as fast as possible. For such situations, take a look at the tooling support available, as well as the ability to do things such as generating database tables from metadata."

"One consideration many don't take into account is whether or not a project provides any best practices; coding standards, project organization recommendations, etc. These are useful to have when you consider long-term maintenance of your project."

"And finally, frameworks tend to focus on specific, core functionality, and leave extensions and support to the community - so you want to look for a vibrant community surrounding the project. This is another area where standards play a role - if the project does not have well-developed standards, often the third-party code will be difficult to evaluate, and you can end up with wide variations in code quality. So, look for a combination of good standards and good community. When you look at community, don't just consider the size of the community, but the quality of the community. Some projects I've looked at do not have a huge following, but the community surrounding them is very knowledgeable and helpful - which makes the framework a pleasure to use. (Solar specifically comes to mind here)."

These are all useful criteria when selecting a framework, but what I like most are Matthew's last words on this question: "Frameworks are like clothes: everybody has preferences on what they like to wear, what looks good, and how to put them together. What's most important is choosing the right clothes for the right situation – or the right framework for the right job. Carefully consider your needs, your organization's needs, and the application needs to make sure you get the right fit."

Matthew mentioned communities and their importance and that users of frameworks are just as important to that community: "If you use a framework, consider contributing back. A framework is only as good as the contributions it receives. If you encounter bugs, write test cases and patches. If you code new features, contribute them back so others may benefit. This helps the entire ecosystem, and prevents others from needless coding. And if you ask questions in the public forum (mailing lists, irc, etc.), answer at least as many, once you have some experience."

The final topic we discussed was the rise of frameworks and the demise of the CMS. 3 years ago, everybody was using one of the popular CMS solutions; nowadays frameworks have taken their place as the preferred choice for building sites and applications. I asked Matthew, "A CMS or a framework?". Matthew: "A framework, hands down. CMS solutions, quite honestly, all suck. CMS solutions do not take into account the varying needs of the organizations utilizing them. Frameworks, on the other hand, allow you to code very specific solutions for your CMS needs. Ultimately, a CMS should allow you to build out functionality as needed, complete with administration interfaces and custom front ends, with a minimum of effort. Frameworks typically provide both, whereas off-the-shelf CMS solutions make custom functionality and output very difficult to achieve."

I concluded the interview by seizing the opportunity to ask Matthew about cool new features he is planning for Zend Framework. Matthew: "Features in ZF 1.8 that will be cool: We're moving towards an autoloading solution, but have to gradually move there due to backward-compatibility issues. Zend_Loader_Autoloader and Zend_Application_Module_Autoloader give more flexibility over autoloading than we've had before, and also provide autoloading capabilities for resources in application modules. This allows for some incredible solutions in regards to lazy loading of resources, as well as inter-module usage of resources. Zend_Application will provide a unified bootstrap, with support for different environments (think development, vs. staging, vs. production), as well as re-use in testing and service infrastructures. I, unfortunately, cannot speak towards 2.0 right now. We're doing a fair bit of planning, but right now a lot of it is up in the air, including if/how we will support PHP 5.3."

IVO JANSCH is the CTO of Ibuildings, a PHP services company based in Europe. He is the author of phparchitect's *Guide to Enterprise PHP Development*, and is an active blogger and speaker in the PHP community. Ivo also initiated the ATK Business Framework.

Cleaning Up

by Derick Rethans

In this second part of the three-part column on the new garbage collecting mechanism in PHP 5.3, we'll dive into a solution to the problem with circular references. If we look quickly back, we found that by using code like is found in Figure 1, an in-request memory leak is created.

Traditionally, reference counting memory mechanisms, such as that used by PHP, fail to address those circular reference memory leaks. Back in 2007, while looking into this issue, I was pointed to a paper by David F. Bacon and V.T. Rajan titled "Concurrent Cycle Collection in Reference Counted Systems" (see Related Links). Although the paper was written with Java in mind, I started to play around with it to see if it was feasible to implement the synchronous algorithm, as outlined in the paper, in PHP. At that moment, I didn't have a lot of time, but along came the Google Summer of Code (see Related Links), and we put forward the implementation of this paper as one of our ideas. Yiduo (David) Wang picked up this idea and started hacking on the first version as part of the Summer of Code project.

A full explanation of how the algorithm works would be slightly beyond the scope of this column, but I will try to explain the basics. First of all, we have to establish a few ground rules. If a refcount is increased, it's still in use and therefore, not garbage. If the refcount is decreased and hits zero, the zval can be freed. This means that garbage cycles can only be created when a refcount argument is decreased to a non-zero value. Secondly, in a garbage cycle, it is possible to discover which parts are garbage by checking whether it is possible to decrease their refcount by one, and then checking which of the zvals have a refcount of zero.

To avoid having to call the checking of garbage cycles with every possible decrease of a refcount, the algorithm instead puts all possible roots (zvals) in the "root buffer" (marking them "purple"). It also makes sure that each possible garbage root ends up in the buffer only once. Only when the root buffer is full does the collection mechanism start for all the different zvals inside. Figure 2 shows this in step A.

In step B, the algorithm runs a depth-first search on all possible roots to decrease by one the refcounts of each zval it finds, making sure not to decrease a refcount on the same zval twice (by marking them as "grey"). In step C, the algorithm again runs a depth-first search from each root node, to check the refcount of each zval again. If it finds that the refcount is zero, the zval is marked "white" (blue in the figure). If it's larger than zero, it reverts the decreasing of the refcount by one with a depth-first search from that point on, and they are marked "black" again. In the last step (D), the algorithm walks over the root buffer removing the zval roots from there, and meanwhile, checks which zvals have been marked "white" in the previous step. Every zval marked as "white" will be freed.

Now that you have a basic understanding of how the algorithm works, we will look back at how this integrates with PHP. By default, PHP's garbage collector is turned on. There is, however, a **php.ini** setting that allows you to change this: **zend.enable_gc**.

When the garbage collector is turned on, the cycle-finding algorithm as described above is executed whenever the root buffer runs full. The root buffer has a fixed size of 10,000 possible roots (although you can

REQUIREMENTS

PHP: *5.3+*

Useful/Related Links:

- David F. Bacon and V.T. Rajan's paper: http://www.research.ibm.com/people/d/dfb/papers/Bacon01Concurrent.pdf
- Google Summer of Code: http://code.google.com/soc/2007/

alter this by changing the **GC_ROOT_BUFFER_MAX_ENTRIES** constant in **Zend/zend_gc.c** in the PHP source code, and re-compiling PHP). When the garbage collector is turned off, the cycle-finding algorithm will never run. However, possible roots will always be recorded in the root buffer, no matter whether the garbage collection mechanism has been activated with this configuration setting.

If the root buffer becomes full with possible roots while the garbage collection mechanism is turned off, further possible roots will simply not be recorded. Those possible roots that are not recorded will never be analyzed by the algorithm. If they were part of a circular reference cycle, they would never be cleaned up and would create a memory leak.

The reason why possible roots are recorded even if the mechanism has been disabled is because it's faster to record possible roots than to have to check whether the mechanism is turned on every time a possible root could be found. The garbage collection and analysis mechanism itself, however, can take a considerable amount of time.

Besides changing the **zend.enable_gc** configuration setting, it is also possible to turn the garbage collecting mechanism on and off by calling **gc_enable()** or **gc_disable()** respectively. Calling those functions has the same effect as turning on or off the mechanism with the configuration setting. It is also possible to force the collection of cycles even if the possible root buffer is not full yet. For this, you can use the **gc_collect_cycles()** function. This function will return how many cycles were collected by the algorithm.

The rationale behind the ability to turn the mechanism on and off, and to initiate cycle collection yourself, is that some parts of your application could be highly time-sensitive. In those cases, you might not want the garbage collection mechanism to kick in. Of course, by turning off the garbage collection for certain parts of your application, you do risk creating memory leaks because some possible roots might not fit into the limited root buffer. Therefore, it is probably wise to call **gc_collect_cycles()** just before you call **gc_disable()** to free up the memory that could be lost through possible roots that are already recorded in the root buffer. This then leaves an empty buffer so that there is more space for storing possible roots while the cycle collecting mechanism is turned off.

In this installment, we saw how the garbage collection mechanism works and how it is integrated into PHP. In the third and final part of the series, we will look at performance considerations and benchmarks.

FIGURE 1

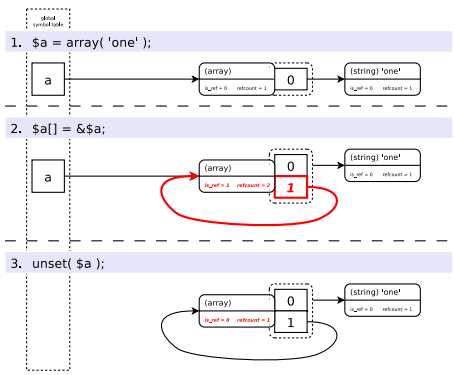
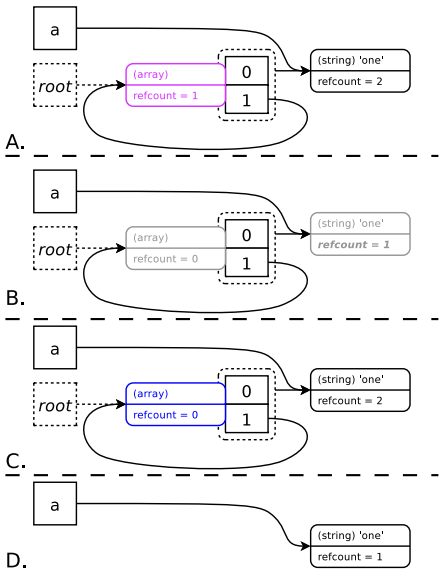


FIGURE 2



DERICK RETHANS has contributed in a number of ways to the PHP project, including the mcrypt, date and input-filter extensions, bug fixes, additions and leading the QA team. He's a frequent lecturer at conferences, the author of phparchitect's Guide to Date/Time Handling, and the co-author of PHP 5 Power Programming. Derick now works as project leader for the eZ components project for eZ systems A.S. In his spare time he likes to travel, hike, ski and practice photography. You can reach him at derick@derickrethans.nl

Installing PHP on Windows

by Hank Janssen and Pierre A. Joye

Many PHP programmers use PHP on Windows today, but most of the core PHP developers work on *nix. As a result, the Windows version of PHP has always lagged behind the other platforms in terms of features and stability. Fortunately, the situation has improved drastically during the last six or so months, with PHP on Windows reaching feature parity with other platforms. Developing with PHP on Windows is now easier than ever and, in some cases, has resulted in important PHP innovations. This article explains some of the changes that have been made in PHP 5.3 for Windows and how you can set up a development environment to compile and develop PHP on Windows within minutes.

Looking Back

PHP has been supported on Windows for quite a few years now, even as far back as PHP 3. In fact, it was more than 10 years ago that PHP received an Internet Information Server interface, but PHP Support for Windows has never been as good as on *nix platforms. For the history fans, there is a PHP museum for all releases (source or binaries) at http://museum.php.net.

Related URLs

PHP: *5.x*

Other Software:

- Windows XP SP2 or later (vista, 2000, server 2003, 2008 or win7)
- Visual c++ 2008 (any version) (free version available here: http://www.microsoft.com/express/download/)
- Microsoft Platform SDK 6.1 - http://www.microsoft.com/downloads/details.aspx?FamilyID=e6e1c3df-a74f-4207-8586-711ebe331cdc&displaylang=en

Optional

- CVS client (for example http://www.tortoisecvs.org)
- DAEMON Tools Lite (http://www.daemon-tools.cc) or any other virtual drive software

Useful/Related Links:

- http://windows.php.net
- http://windows.php.net/downloads
- http://wiki.php.net/internals/windows
- http://pecl2.php.net/downloads/php-windows-builds/php-libs/binary-tools.zip
- http://pecl2.php.net/downloads/php-windows-builds/php-libs/

Support Section:

IRC Channel: Freenode #php-dev-win

Windows Internals mailing list: http://www.php.net/mailing-lists.php

Compiling PHP used to be an adventure of epic proportion. Build problems, dependency issues and non-compatible libraries were just some of the challenges, and only the hardy prevailed in completing a build of PHP for Windows. The various dependencies were not ported to Windows, and for those that were, finding working ports was sometimes even harder than building PHP. In fact, many of the libraries used to build PHP 5.2 or older were last built in 1997, and the source code that was used for them was lost to the ages.

Only a single developer was taking care of the Windows builds, snapshots and releases, and no clear build instructions existed. Things became even more complicated when somebody had to reproduce the builds using the exact same configuration to create binary-compatible extensions or to simply try to update a dependency (library).

Many of the critical PHP features depend on external libraries. The use of these libraries greatly helps PHP provide the incredible rich set of features it has today. These libraries are available on every major distribution and are frequently updated. For Windows, we have to do extra work for these libraries, as 90% of them do not provide compatible binaries or in some cases do not provide binaries at all.

And this is where the real trouble began; very few libraries were kept up-to-date. It was impossible to trace which sources were used to build them, which version or patches were used. Additionally, many new features were simply not available on Windows.

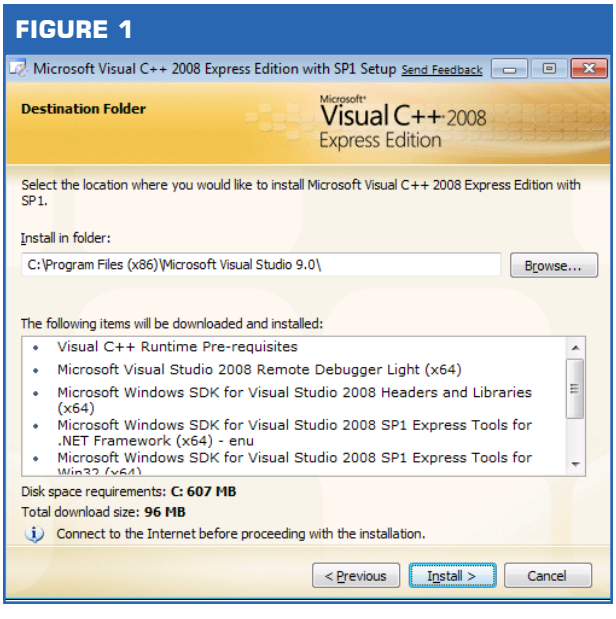
Another issue was the compiler that was used to build PHP, VC 6 (the compiler that came with Visual Studio 1998): a 10 year old compiler for which you could no long buy licenses and which was no longer supported by Microsoft.

PHP on Windows was slowly dying due to lack of support and community members interested enough to want to work on these issues.

This situation persisted until the PHP 5.2.6 release time (April/May 2008), but with an ever-increasing interest in Windows as a platform for PHP deployments, it had to change. In April 2008, the PHP community, with the help of Microsoft's Open Source Center, decided to drastically improve this situation, and the PHP Windows Team was born. Work began on improving the next PHP (5.3) on Windows.

PHP 5.3 introduced a completely new build system for Windows. Most of the project files and the associated pain went away. The new system is made up of a set of Jscript scripts reproducing the syntax of the m4 configuration file available on the other platforms (the config.m4 files or the main configure command). The new scripts also are easy to extend and maintain. PHP 5.3 and its new build scripts brought PHP on Windows to a new level, opening the door for many features and extensions available on other platforms to be ported to Windows.

FIGURE 1



Where We Stand

One of the first tasks was to make sure PHP could be built smoothly with a recent compiler. It was virtually impossible to debug PHP on Windows with a 10+ year old compiler, with much of the source no longer available.

We choose to target Visual Studio 2008 (referred to as VC9 in this article). Visual Studio 6 (VC6) is still supported for PHP Builds, and the current plan is to provide VC6 support for the life of the 5.3 Branch. Doing so will allow third parties to migrate to VC9 as well. It is important to note that the community is planning for any release after 5.3 to be available only as VC9 binaries.

VC9 is, at the writing of this article, the latest version of the Microsoft Visual C++ environment. It is available in many different versions. Yes, MSFT product managers love to confuse you with weird naming, but shh, don't tell them we said so. :) Any versions of VC9 will be able to compile PHP, even the free (as in beer) Visual C++ Express. You can download that version at http://www.microsoft.com/express/download/.

Another very nice side effect of all this work is that the community now has both 32-bit and 64-bit versions of PHP on Windows with VC9. Using VC9 gives us a much improved compiler over VC6, and it will allow us to now further optimize the core PHP system. For example, the **bench.php** script (available in the PHP sources, in **Zend/bench.php**) takes 7.65 seconds to be executed using a VC6 build while it takes only 4.34 seconds using a VC9 build. A 40% performance improvement! Not bad for just replacing the compiler!

The second task, which is still ongoing, concerns the PHP libraries used by PHP and its extensions. 80% of the dependencies are now **traceable**, **documented** and ported to Windows, for both VC6 and VC9. A complete list is available on the PHP community wiki as well as the procedure to compile each of them. See http://wiki.php.net/internals/windows/libs. This list includes each PHP version's specific details and is kept updated with what we actually use on the official PHP build machines (releases or snapshots).

While in the process of cleaning up the PHP source code from all Windows-specific hacks and patches, we have succeeded in dropping all dependencies for the core PHP language. It means that it is now possible to compile PHP without having to fetch external libraries; only a compiler and the Windows Platform SDK is necessary to build PHP.

PHP can be compiled with VC6 or VC9 (other version are known to work but are not officially supported), but we will focus on VC9 only in this article. A list of all compilers and their respective Platform SDK can be found on our wiki, see http://wiki.php.net/internals/windows/compiler and http://wiki.php.net/internals/windows/windowssdk.

As a result of the ongoing work, the community can now support a wider range of Windows versions, even the latest new releases like the Windows 7 beta:

Officially supported:

- Windows 2008/32bit
- Windows Vista/32bit
- Windows 2003/32bit
- Windows 2000/32bit
- Windows XP SP2/3 32bit

Experimental:

- Windows 2003/64bit
- Windows 2008/64bit
- Windows Vista/64bit
- Windows 7/32 or 64bit

Please note that Windows 64bit can run 32bit binaries without any problems.

So, now enough of the boring background! Let's actually build PHP on Windows!!

Setting Up Your Environment

Any of the above listed Windows versions should work. We actually used the public beta of Windows 7 for this article. Before you start, make sure you have enough free space to install the different tools and components required to compile PHP. We usually try to have five to six or so GB of disk space free before we do a clean install for the build environment. Visual Studio Express is not very large but with all the libraries, tools, sources as well as the temporary object files created during the build process, we will use up a fair amount of space, and the platform SDK requires 3GB for the install process.

FIGURE 2

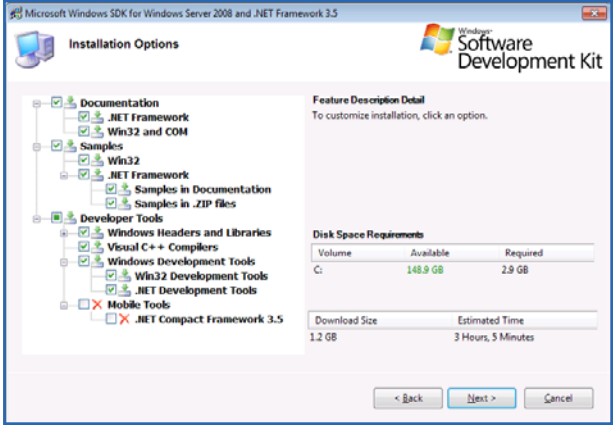
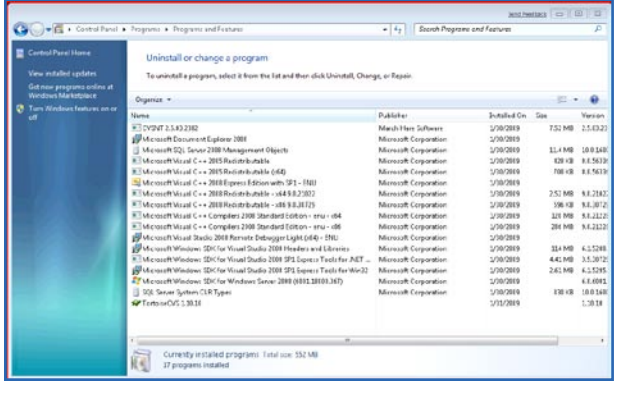


FIGURE 3

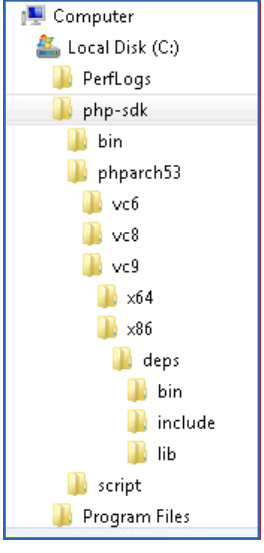


The first large download is the VC9 compiler (Visual C++ 2008). If you have not downloaded it yet, fetch it here http://www.microsoft.com/express/download/. If you plan to use the compiler on non network-connected machines or if you like to have a CD at hand, you can choose the offline install option, using the ISO download. It can be found at the bottom of the page, in the **Offline Install** box.

Use a tool like Daemon Tools Lite to mount the iso and run the setup or simply use the web installer. Figure 1 shows the last form before the install begins, using the web installer. You should see similar options.

The second download is the Platform SDK 6.1, which can be found here http://www.microsoft.com/downloads/details.aspx?FamilyID=e6e1c3df-a74f-4207-8586-711ebe331cdc&displaylang=en. Figure 2 shows the necessary options.

FIGURE 4



If you like to work with the latest version of PHP 5.3, we suggest you install a CVS client as well, for example TortoiseCVS, available from http://www.tortoisecvs.org/. Really, the best PHP for Windows improvements are in PHP 5.3 or newer.

Once you are done with the installation of these packages, you should see something like the window in Figure 3.

Finally, we need a few more tools which are used by the PHP build scripts, (re2c, bison, zip, etc.) as well as a couple useful commands to set up a development directory structure. It is the basics of what we called the **PHP SDK**. You can find it here: http://pecl2.php.net/downloads/php-windows-builds/php-libs/binary-tools.zip. Uncompress it in the location where you would like to have the PHP SDK. We chose **C:\php-sdk** for this article.

You can now open the Windows SDK Shell. We will target Windows XP/32bit as the version for our binaries. To set the correct environment for this platform, enter the following command in the freshly-opened shell: **setenv /x86 /xp /debug**

We are also choosing to turn on the debug mode. We will need it later to debug PHP or its extensions. Now go to the php-sdk: **cd c:\php-sdk**

The first thing to do is to set the various PHP-SDK path, by running: **bin\phpsdk_setvars.bat**

Now, all PHP SDK scripts and commands can be found in your path. You have to run this command after every call of the **setenv** command. **setenv** overrides the PATH environment variable. To avoid confusion and problems due to incompatible binaries (the various Visual C++ runtime CRT are not compatible with each other), we defined a well-separated directory structure. Each compiler version and architecture (for example VC9 and x64) has its own directory. It may sound like overkill if you use only one compiler but doing so will also ease the debugging process. Having a standard directory structure is a great help during problem-hunting via our mailing list or on IRC (see the Support Section). There is a command in the PHP SDK to create such a structure. It simply takes the name of the top directory (the branch name for example). We named ours **phparch53**: **phpsdk_buildtree phparch53** You should see the structure in Figure 4.

Now, you can fetch the PHP 5.3 sources. The PHP Windows site provides hourly snapshots of the PHP 5.3 sources. They can be fetched on the snapshots page http://windows.php.net/snapshots/. Alternatively, you can use CVS to fetch the latest version of PHP 5.3. For completeness, and as you will surely need CVS later, CVS will be used here.

Open a file explorer (Windows key + E) and go

FIGURE 5

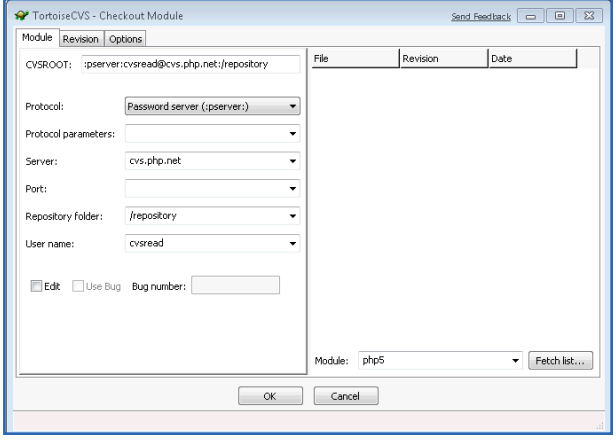


FIGURE 6

to c:php-sdkphparch53vc9x86. Right click on the **x86** folder and select the **CVS Checkout ...** command. You should see the dialog shown in Figure 5. Copy the following line in the CVSROOT field: **:pserver:cvsread@cvs.php.net:/repository** then enter **php5** in the module field (bottom right of the window). Next, click on the **Revision** tab and enter **PHP_5_3** in the **Branch or tag name** (Figure 6).

php5 is a module alias for all PHP 5.x branches. Once you feel comfortable enough, you may want to play with php6. **PHP_5_3** is the name of the PHP 5.3 development branch. For PHP 6, you can leave this field blank; it is commonly called HEAD.

Alternatively, you can use the console CVS tools (also installed via tortoiseCVS or using the binaries from http://cvshome.org). The procedure is described in detail at http://www.php.net/anoncvs.php.

We hope you are still with us, the painful part is done now!

Your First Build

Now you have everything you need, so you can finally build PHP 5.3! The features requiring external libraries are not yet available, but all the core functionality, as well as many widely used extensions, can be built already.

Go to the php source directory: **cd phparch53\vc9\x86\php5** and generate the configure script: **buildconf**

If everything goes well, you should see the following text:

```
C:\php-sdk\phparch53\vc9\x86\php5>buildconf

Rebuilding configure.js
```

Now run **configure -help**

Finally, you can configure php. The options used here will let you compile a tiny php:

```
configure --disable-all --enable-cli --enable-debug
```

This command tells the configure script to disable everything possible except the CLI sapi using the debug mode. A summary table lists all enabled extensions as well as the options for the current build (thread safety, debug/release mode, etc.), see Listing 1. Now you can compile PHP using: **nmake**

You should see this on success (if it fails at this point, you probably have a configuration problem):

```
Microsoft (R) Windows (R) Resource Compiler Version
6.0.5724.0 Copyright (C) Microsoft Corporation.  All
rights reserved. SAPI sapi\cli build complete
```

To verify the build, you can list the PHP extensions available:

```
C:\php-sdk\phparch53\vc9\x86\php5>Debug_TS\php.exe
-m

[PHP Modules]
Core
date
ereg
pcre
Reflection
SPL
standard

[Zend Modules]
```

It is a scaled-down version of PHP, not very useful yet, especially if you like to use databases or other more advanced features, so let's activate more options:

FIGURE 7

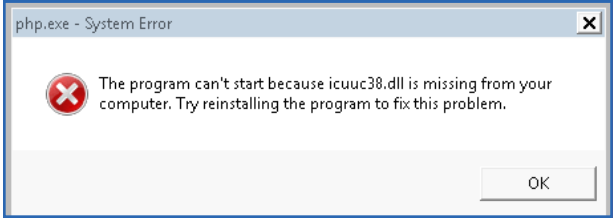


FIGURE 8

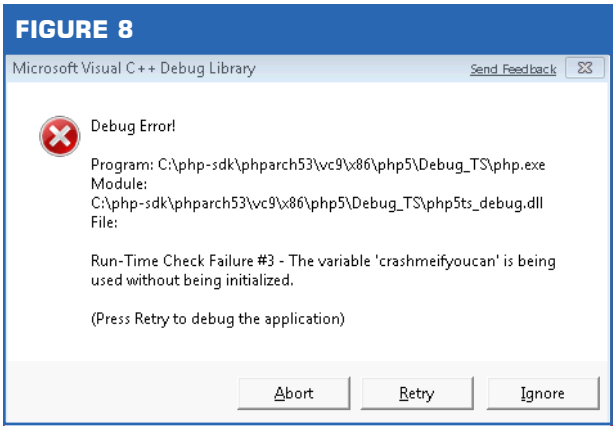
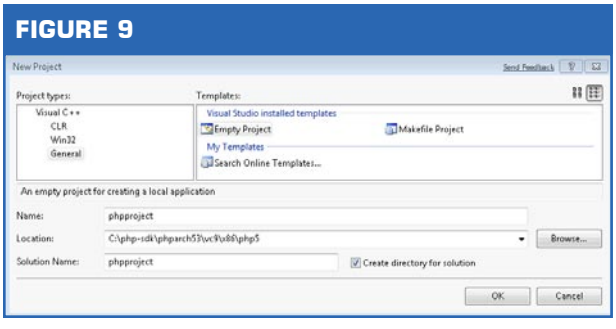


FIGURE 9



```
C:\php-sdk\phparch53\vc9\x86\php5>configure.bat
  --disable-all --enable-cli --enable-debug
  --with-sqlite3 --enable-pdo --with-pdo-mysql
  --with-pdo-sqlite --enable-json
```

This will show:

```
Enabled extensions:
-----------------------   -----------------------
| Extension   | Mode    |   | Extension   | Mode    |
-----------------------   -----------------------
| date        | static  |   | pdo_sqlite  | static  |
| ereg        | static  |   | reflection  | static  |
| json        | static  |   | spl         | static  |
| pcre        | static  |   | sqlite3     | static  |
| pdo         | static  |   | standard    | static  |
| pdo_mysql   | static  |   -----------------------
-----------------------
```

There is also a mode to tell the configure script to enable everything it has. This mode is called the snapshot build:

```
configure.bat --enable-snapshot-build --enable-
debug''
```

As you can see below, you are now getting a nice set of features. More than thirty extensions are enabled and still no single external dependency.

```
Enabled extensions:
-----------------------   -----------------------
| Extension   | Mode    |   | Extension   | Mode    |
-----------------------   -----------------------
| bcmath      | static  |   | odbc        | static  |
| calendar    | static  |   | pcre        | static  |
| com_dotnet  | static  |   | pdo         | static  |
| ctype       | static  |   | pdo_mysql   | shared  |
| date        | static  |   | pdo_odbc    | shared  |
| ereg        | static  |   | pdo_sqlite  | shared  |
| exif        | shared  |   | phar        | static  |
| filter      | static  |   | reflection  | static  |
| ftp         | static  |   | session     | static  |
| hash        | static  |   | shmop       | shared  |
| json        | static  |   | sockets     | shared  |
| mbstring    | shared  |   | spl         | static  |
| mysql       | shared  |   | sqlite      | shared  |
| mysqli      | shared  |   | sqlite3     | shared  |
| mysqlnd     | static  |   | standard    | static  |
-----------------------   | tokenizer   | static  |
                          -----------------------

Enabled SAPI:
-------------
| Sapi Name |
-------------
| cgi       |
| cli       |
| cli_win32 |
| embed     |
| isapi     |
-------------
```

Configure PHP and Its Dependencies

Now that you are ready to compile PHP, it is time to use some of the external libraries. It is very straightforward to add any libraries to your development tree.

Each architecture tree (vc9/x86 for example) contains a **deps** directory. It is used to store all files used to build or run PHP. It is comparable to the usr or opt directories, if you are familiar with a *nix platform. It is composed of three sub directories (see Figure 4 again):

- bin: contains all DLLs or libraries specific binaries
- include: contains all headers file (.h, .hpp, etc.)
- lib: contains all development libraries used to link with php

This directory is automatically detected by configure and is treated as the default location for the files of each library.

All libraries are available online and can be downloaded for each compiler and architecture combination. The base URL for the downloads is http://pecl2.php.net/downloads/php-windows-builds/php-libs/. Since the same directory structure is used as in our PHP SDK directories, it is very easy to find the one you need. Our current configuration being VC9 and x86, you can find all libraries at http://pecl2.php.net/downloads/php-windows-builds/php-libs/VC9/x86/.

Each of the library archives contains the **bin**, **include** and **lib** directories. To add a library to your development, all you have to do is copy the archive contents (bin, include and lib) to the **deps** directory.

For example, let us try to enable the hot new **intl** extension (a cool new extension providing I18N and some unicode features to php 5.3). First, you have to know which libraries are required by the **intl** extension. You can do this by checking the ever-useful Wiki. It has a list of all extensions and their dependencies, including the version info. See http://wiki.php.net/internals/windows/libs#extensions_and_their_libraries.

As you can see, the **intl** extension depends on the ICU library. The version used by PHP 5.3 on VC9/x86 is 3.8.1 (it could be another minor version like 3.8.2 by the time this article is published). Download the corresponding zip archive and extract its content to the **deps** directory.

Go back to the SDK shell, and execute: **config.nice**

This command calls the configure script using the same options as in the last configure call:

```
configure.bat --enable-snapshot-build --enable-
debug''
```

You should see the **intl** extension in the summary table:

```
Enabled extensions:
-----------------------
| Extension   | Mode   |
-----------------------
| bcmath      | static |
...
| hash        | static |
| intl        | shared |
| json        | static |
...
| tokenizer   | static |
-----------------------
```

Run **nmake** to compile a new PHP.

Now let's see if our new shared extension can be loaded:

```
Debug_TS\php.exe -dextension_dir=Debug_ts
-dextension=php_intl.dll -m''
```

And it fails miserably with a nice dialog (see Figure 7) and the following message:

```
PHP Warning:  PHP Startup: Unable to load dynamic
library \'Debug_ts\php_intl.dll\' - The specified
module could not be found. in Unknown on line 0''
```

What went wrong?

The ICU libraries are DLLs and are not statically linked (like being entirely copied into the intl extension) to PHP or to **intl**. They remain separate and have to be placed in a directory where the system can find them. In our case, they can be found in the **deps** directory, in **bin**. To fix this error, you simply need to add this **bin** directory to the PATH environment variable, first in the list (to be sure that a system version cannot be used instead of the SDK one):

```
set PATH=C:\php-sdk\phparch53\vc9\x86\deps\
bin;%PATH%''
```

and now call again:

```
Debug_TS\php.exe -dextension_dir=Debug_ts
-dextension=php_intl.dll -m''
```

Et voila, you can now begin to test your freshly built PHP and even add any other extension you may need.

Debugging PHP, the Easy Way

There is no fun without bugs right? Bugs are part of a developer's life, just like Monday mornings. As we all know, PHP has no (known) bugs, so we have to create one for our little exercise.

Open the file C:php-sdkphparch53vc9x86php5extstandardinfo.c and add the code in Listing 2 inside the phpinfo function (line 1205). Run nmake again and call php using: **Debug_TS\php.exe -r "phpinfo();"**

As to be expected, it crashed and Windows is asking you how you would like to proceed, as seen in Figure 8. If you are a lucky user of Visual Studio Standard, Pro or Team edition, you can skip the following steps, as these versions support a just-in-time debugger. You do not need to attach the debugger to the crashed process. If you use Visual Studio Express, the **attach process** procedure described below is a necessary step.

Do not click any button in the dialog (Figure 8), but instead open Visual C++ Express (Windows > Programs > Microsoft Visual C++ 2008 Express Edition). Create a new empty project (File > new > Project) as shown in Figure 9 and save it in the php source directory (its location is not important). The project will remain empty, it is only a dummy project.

Once you are done, you will be able to attach a process to the debugger. Select the Debug menu and click on the **Attach to Process...** command. The dialog shown in Figure 10 will appear. Select **php.exe** and click the **attach** button. Now, go back to the error dialog in Figure 8, and click **retry**. The debugger will then catch the error and let you begin the debugging (see Figure 11) session, exactly where the crash occurred.

You would certainly prefer to have set a breakpoint earlier, at a stage where you know that everything was working. That's possible using the **DebugBreak()** macro. Modify the phpinfo function using the code in Listing 3. Compile PHP again (nmake).

This time you will need a little pause to attach the

LISTING 1

```
Enabled extensions:
-----------------------
| Extension   | Mode   |
-----------------------
| date        | static |
| ereg        | static |
| pcre        | static |
| reflection  | static |
| spl         | static |
| standard    | static |
-----------------------

Enabled SAPI:
-------------
| Sapi Name |
-------------
| cli       |
-------------

-----------------------------------------
|               |                       |
-----------------------------------------
| Build type    | Debug                 |
| Thread Safety | Yes                   |
| Compiler      | MSVC9 (Visual C++ 2008) |
| Architecture  | x86                   |
-----------------------------------------

Type 'nmake' to build PHP
```

debugger to the php.exe before it crashes (JIT debugger again). A sleep(10) will give you ten precious seconds to do it, as described earlier (increase the time if necessary and open the **Attach to Process** dialog before): **Debug_TS\php.exe -r "sleep(10);phpinfo();"** You will end up directly in the debugger, at the location of the breakpoint.

The same can be done using dynamic breakpoints, by clicking on the gray margin on the right side of the source editor. A red dot signals the location of the breakpoint. Click it again to remove it.

One of the things we always ask for when you file a bug on bugs.php.net is the backtrace (or call stack).

LISTING 2

```
1. /* {{{ proto void phpinfo([int what])
2.    Output a page of useful information about PHP and the
3.    current request */
4. PHP_FUNCTION(phpinfo)
5. {
6.   long flag = PHP_INFO_ALL;
7.
8. /* Our bug begins here */
9.   char *crashmeifyoucan;
10.
11.   printf("%s\n", crashmeifyoucan);
12. /* and ends here */
13.
14.   if (zend_parse_parameters(ZEND_NUM_ARGS() TSRMLS_CC,
15.       "|l", &flag) == FAILURE) {
16.     return;
17.   }
18. ...
19. }
```

FIGURE 10

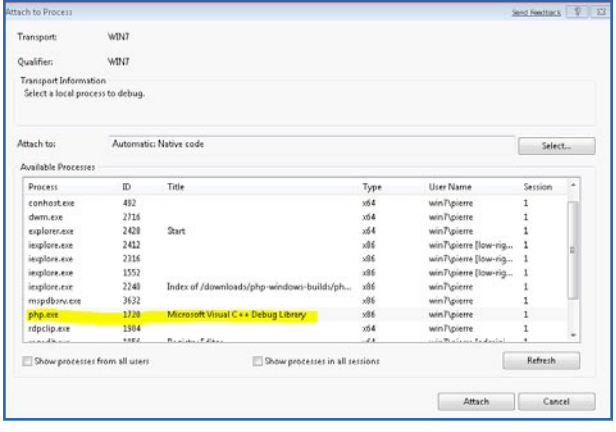
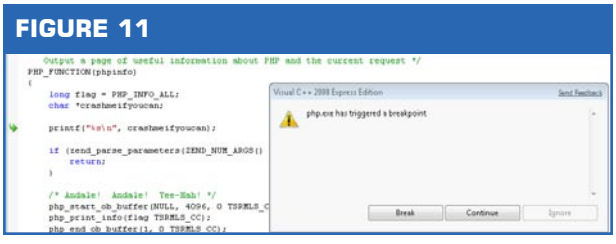


FIGURE 11



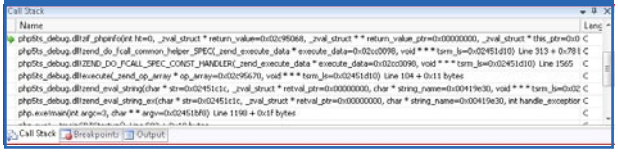
It is always available when you enter a debugging session. It shows you the call tree of your application before it crashed. You can find it at the bottom right of the Visual c++ Express window (see Figure 12). It can be copied as text (select all lines and press ctrl-c), very handy to quickly paste it into a bug report.

Well, congratulations, you now know how to build PHP on Windows and some of the basics of debugging. What used to be the tribal knowledge of one, now has one more person able to build it, namely You!

Going Further and Support

If you have questions or would like to help us improve PHP on Windows, you are more than welcome! You can find our mailing list and IRC channel in the Support Section.

FIGURE 12



LISTING 3

```
1. /* {{{ proto void phpinfo([int what])
2.    Output a page of useful information about PHP and the
3.      current request */
4. PHP_FUNCTION(phpinfo)
5. {
6.   long flag = PHP_INFO_ALL;
7.
8. /* Our bug begins here */
9.   char *crashmeifyoucan;
10.   DebugBreak();
11.   printf("%s\n", crashmeifyoucan);
12. /* and ends here */
13. ...
14. }
```

HANK JANSSEN is the Director of Program Management at the Microsoft Open Source Technology Center where he runs the Open Source Software Lab and Novell Joint Interoperability labs. Hank has been working with *nix and other OSS for over 20 years. He started at AT&T doing kernel programming for the SYS V process scheduler for Digital Telephone switches (5ESS). He designed a globally distributed database system using such golden oldies as uucp and rje. He was also the lead programmer/designer on a Point Of Sale system that did real-time cell phone activations for Sprint - the first in the US - built solely with OSS software. Favorite languages remain C/C++ and his favorite editor is still Emacs. :)

Introduction to Silverlight and the HTML DOM

by Joe Stagner

This article is intended to serve as an introduction to Microsoft Silverlight for developers that are probably not .NET developers. There is an AMAZING quantity of speculative dialog on the world wide web about what Silverlight is, what it's good at (or not), what it means to developers and designers, and especially whether or not it's a "Flash Killer." Even an introduction to Silverlight would be incomplete without some discussion of WHY Silverlight. Is Silverlight a Flash Killer? This question presumes that Silverlight and Flash are designed to meet the same set of development needs, and I think this assumption is an incorrect one. Sure, they're both "RIA" technologies, whatever that means these days, but there is a much broader story to tell when comparing the two.

Requirements

- Visual Studio 2008 or Visual Studio Web Developer Express from Microsoft.com http://www.microsoft.com/express/vwd/
- Silverlight tools for Visual Studio which are available here: http://silverlight.net/GetStarted

Microsoft with Silverlight and Adobe with Flash each have diametrically opposed problems when it comes to the adopting audiences. Microsoft really "gets" the Developer, but is relatively new at embracing the Designer community and mindset. Adobe really "gets" the Designer, but is still learning to embrace the Developer community and mindset. I'm not an artistic guy. In fact, I'm what is commonly known as "Artistically Challenged", so for me, comparative discussions like the relative merits of frame-based animations versus timeline-based animations are completely meaningless.

Sure, Silverlight, combined with the Expression Studio Tools, and a person with visual talents can create great animations and other visual effects using the vector graphics system, and sure, Silverlight has a great Media (Audio / Video) story, but you have to think a little harder to really understand the doors that Silverlight is going to open. Silverlight 2 is .NET. This means a couple of important things. First, it means .NET languages. From Microsoft, it means first class language support in the form of C# and Visual Basic and dynamic languages like managed JavaScript, IronPython and IronRuby as well as the syntax varieties that are implemented by any other language vender. Second, it means that a significant subset of the .NET Framework becomes available to code that will be running on the client, in the browser. Here are some highlights:

- WPF UI Framework: Silverlight 2 includes a rich WPF-based UI framework that makes building rich Web applications much easier. In includes a powerful graphics and animation engine, as well as rich support for higher-level UI capabilities like controls, layout management, data-binding, styles, and template skinning. The WPF UI framework in Silverlight is a subset of the WPF UI framework features in the full .NET Framework, and enables developers to reuse skills, controls, code and content to build both rich cross-browser web applications, as well as rich desktop Windows applications.

- Rich Controls: Silverlight 2 includes a rich set of built-in controls that developers and designers can use to quickly build applications and includes core form controls (textbox, checkbox, radio button, etc), built-in layout management panels (StackPanel, Grid, Panel, etc), common functionality controls (Slider, ScrollViewer, Calendar, DatePicker, etc.), and data manipulation controls (DataGrid, ListBox, etc). The built-in controls support a rich control-styling and templating model, which enables developers and designers to collaborate together to build highly polished solutions.

- Rich Networking Support: Silverlight 2 includes rich networking support. It includes out of the box support for calling REST style, SOAP, POX, RSS, and standard HTTP services. It supports cross-domain network access (enabling Silverlight clients to directly access resources and data from resources on the web). There is also built-in socket networking support.

- Rich Base Class Library: Silverlight 2 includes a rich .NET base class library of functionality (collections, IO, generics, threading, globalization, XML, local storage, etc.). It includes rich APIs that enable HTML DOM/JavaScript integration with .NET code. It also includes LINQ and LINQ to XML library support (enabling easy transformation and querying of data), as well as local data caching and storage support. The .NET APIs in Silverlight are a compatible subset of the full .NET Framework.

This partial feature set is compelling, but how about a bit of context for what some of these features might mean. The advanced language constructs mean not having to make JavaScript jump through hoops when implementing REAL business logic. Because our server-side code and our client-side code can both be written in C# or Visual Basic and that code will run at "compiled code" speeds means we no longer need to avoid heavy computational logic running client-side.

We can even envision things like a rules engine or other business logic in a single source code tree running either client-side or server-side. Or, we could implement cooperative encryption between our Silverlight client and our server-side code. The WPF UI framework means that we can use the same skill set to develop Silverlight applications and WPF client applications. We can even share some code and UI elements between the two, and the Rich Controls coupled with the ability to create user controls means high levels of UI reuse.

Then, when we start to look at the networking stack and the BCL (Base Class Library) things really start to get interesting. Just to start thinking

about how Silverlight is way more than a UI technology, imagine a browser-based application based on Silverlight that has multiple simultaneous threads of execution. Thread one constantly interacting with the user interface, updating its real estate as new data is available. Thread two works in the background running business logic that selectively retrieves data from different sources on the internet. One is a secure SOAP-based web service, another data source is a simple REST-based endpoint, and yet another is a high-flow data stream that our Silverlight application communicates with via sockets to eliminate the overhead of HTTP and a text-based payload. As the data is retrieved, it is stored locally in client-side isolated storage so that the same data need not be re-downloaded unless it's desirable to do so. And maybe it all happens

FIGURE 1

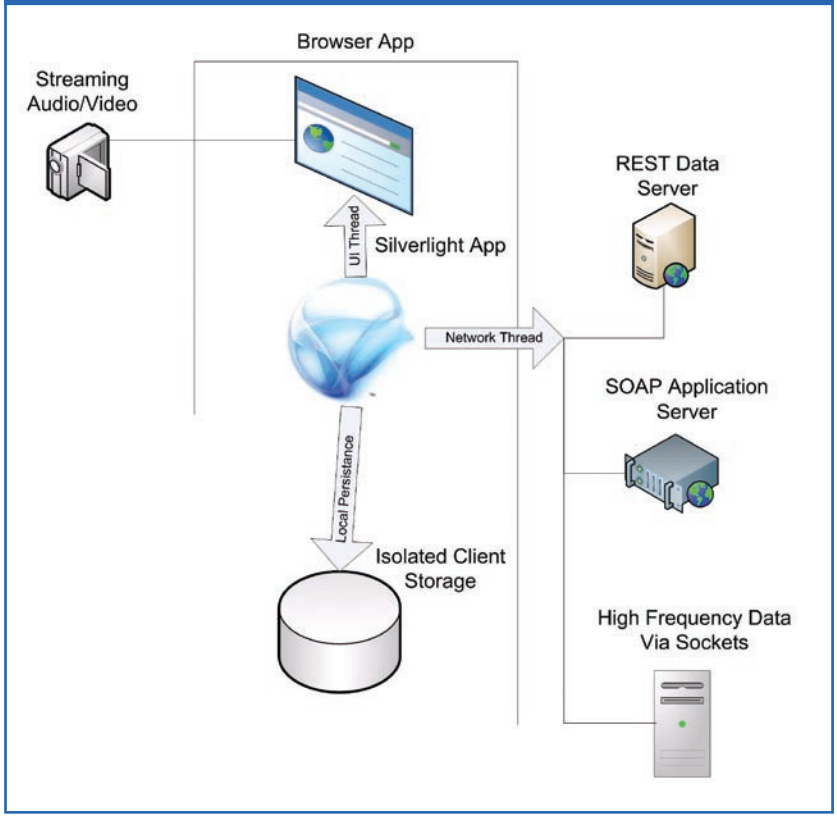
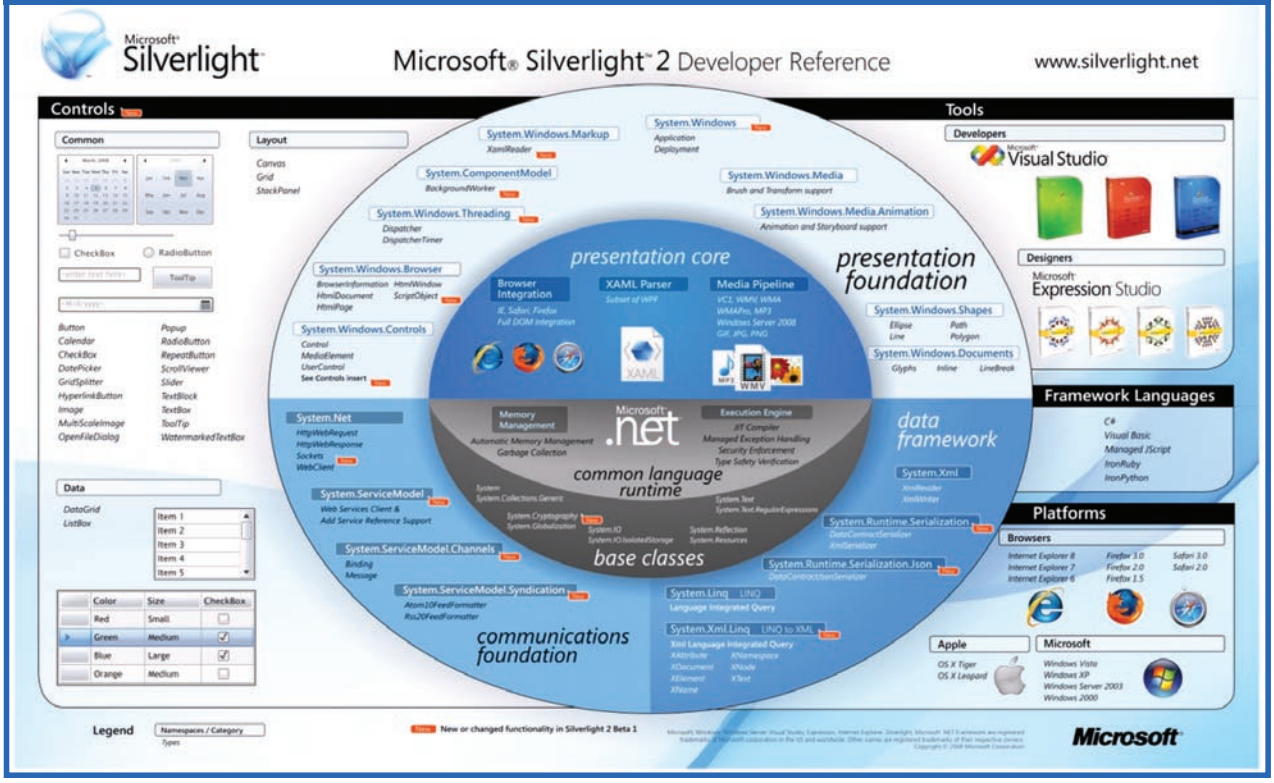


FIGURE 2



while we are watching a streaming video (see Figure 1).

The complete feature set delivered by Silverlight is much larger as the developers diagram illustrates in Figure 2.

Now that we've spent some time thinking about how Silverlight might be used by a client-side developer and have taken a 10,000 foot peek at just some of the functionality that Silverlight contains. Let's move on to some technical specifics.

According to Wikipedia:

> *"Microsoft Silverlight is a programmable web browser plug-in that enables features such as animation, vector graphics and audio-video playback that characterize rich Internet applications. Version 2.0, released October 2008, brings additional interactivity features and support for .NET languages and development tools. It is compatible with multiple web browser products used on Microsoft Windows and Mac OS X operating systems. Mobile devices, starting with Windows Mobile 6 and Symbian (Series 60) phones, will also be supported."*

The vector-based presentation in a Silverlight control or application is powered by the "Extensible Application Markup Language" or XAML for short.

Because XAML is used for visual objects and behaviors in WPF and Silverlight, it's often assumed that XAML is a User Interface Definition Language, but this is not strictly the case.

In a more general sense, XAML is an XML vocabulary for defining collections of objects. These objects can be visual, logical, or behavioral. It just so happens that the first practical applications of XAML are user interface (and UI behavior) related.

To get started building Silverlight applications, you'll need a couple of tools. If you don't already have Visual Studio 2008, you can download Visual Studio Web Developer Express from Microsoft.com (http://www.microsoft.com/express/vwd/) . You will also need the Silverlight tools for Visual Studio which you can download from here: http://silverlight.net/GetStarted.

In Visual Studio 2008, WYSIWYG support is developer focused which is to say that Drag n' Drop works in code and the design view serves as a read-only visual preview. Though I find this fine for working from the developer's perspective, richer WYSIWYG support is planned for a future version of Visual Studio.

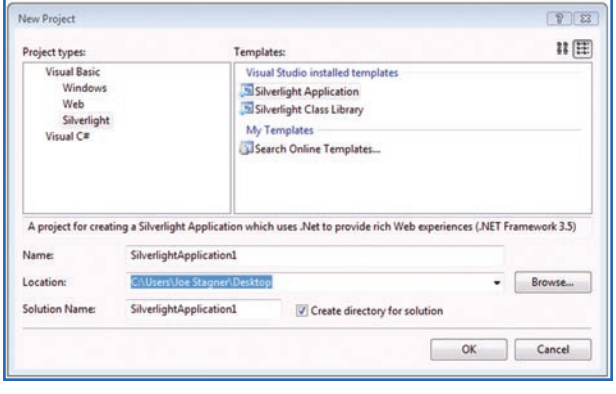
If you're a designer, you may want to check out Microsoft's Expression Blend and the other Expression tools. Expression Studio (which contains the all the Expression products) currently retails for $699. Expression Blend is the product in the suite that a designer would use for creating XAML-based user interfaces that can then be used by developers using Visual Studio as they share a common project file format.

Note that Expression Studio, in addition to Blend includes a graphical design tool (Expression Design), a web development tool (Expression Web) which includes PHP syntax support, Expression Media (for working with Video / Audio) , and Expression Encoder (for encoding media assets).

Once the tools are installed, you are ready to build your first Silverlight application. Though Silverlight controls and applications are built using .NET, PHP developers can integrate with Silverlight applications in two ways. First, the Silverlight runtime includes a complete networking stack so that your Silverlight application can call your PHP code via REST, SOAP, or even Sockets. As PHP developers, we're very familiar with constructing an HTML DOM and delivering JavaScript from our PHP code. You can build a Silverlight application in which your entire user interface is a single Silverlight Canvas, and the browser basically serves as a delivery mechanism. Or, you can build a Silverlight application in which one or more Silverlight objects are contained in an HTML page. In this scenario, one can call compiled code in the Silverlight application from JavaScript code in the containing page. It is also possible to call JavaScript code in the containing page from the compiled code in the Silverlight application. This is the scenario we'll implement below. Visual Web Developer will create the raw Silverlight project as seen in Figure 3.

In addition to the Silverlight project files, Visual Web Developer will also create test pages in ASP.NET

FIGURE 3



and HTML for running our Silverlight application. For the purposes of this article, we'll ignore the .aspx file and work with the .html file which best represents the markup we would generate from a PHP application (see Figure 4).

If you're following along, you can delete the default.aspx and the [YourProjectName]TestPage.aspx files. For simplicity's sake, I'll rename the SIlverlightApplications1TEstPage.html file to index.html (Note that SIlverlightApplication1 is simply the name of my project and it's used in the names of the auto generated files.)

You'll notice a Silverlight.js file in the project and this line in the <head> section of our HTML page:

```
<script type="text/javascript" src="Silverlight.
js"></script>
```

This is the Siverlight bootstrapper. It makes sure the Silverlight runtime is installed on the client and exposes necessary functionality to JavaScript. It's sparsely commented but a quick look at the code will give you the gist of its functionality.

Also defined in your <head> section is the JavaScript function "onSilverlightError" this is a general error catcher for Silverlight, and you can replace it with whatever is appropriate for your application. In the body of our html page is the <object> tag that contains the Silverlight application (see Listing 1).

The parameters for the Silverlight application are pretty self-explanatory but there are a few notable properties. First, note that the "onerror" parameter references the JavaScript function defined in our pages <head> section. Note also, the version checking parameter and the **<param name="minRuntimeVersion" value="2.0.31005.0" />** and the following section that provides version upgrading guidance to the user.

Next, we'll look at the two XAML files that were created when we selected the Silverlight project type and their associated "Code-Behind" files (see Figure 5). The Page.xaml file, is the base user interface of your Silverlight application and is loaded when the application activates. In Page.xaml, we can use UI controls to define our user interface and then handle events from them in the Code-Behind file.

The App.xaml file is typically used to declare resources, such as brush and style objects that are shared across the application. The Code-Behind file for the App.xaml file can be used to handle application level events - like Application_Startup, Application_Exit and Application_UnhandledException. Since our Silverlight application will be used to execute logic and will have no "Silverlight UI / graphics", we'll be working with App.xaml.vb. Note that I just happened to choose Visual Basic, but you can write your Silverlight application in any supported .NET language.

Our first exercise will be to populate an element in the HTML DOM from the compiled code in the Silverlight object when the Silverlight object loads. After adding a textbox element to the page, we can populate it in the Silverlight object's "Application_Startup" event handler:

FIGURE 5

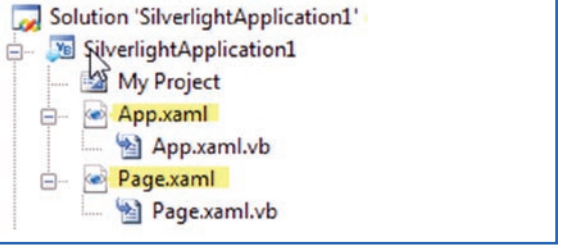
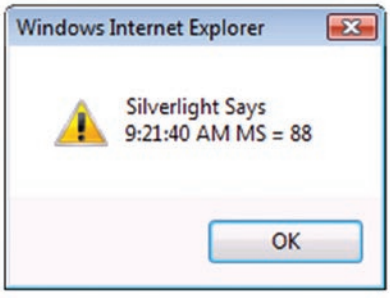


FIGURE 6

FIGURE 7



LISTING 1

```
1. <object data="data:application/x-silverlight-2,"
2. type="application/x-silverlight-2" width="100%"
3. height="100%">
4. <param name="source" value=
5. "ClientBin/SilverlightApplication1.xap" />
6. <param name="onerror" value="onSilverlightError" />
7. <param name="background" value="white" />
8. <param name="minRuntimeVersion" value="2.0.31005.0" />
9. <param name="autoUpgrade" value="true" />
10. <a href="http://go.microsoft.com/fwlink/?LinkID=124807"
11. style="text-decoration: none;"><img src=
12. "http://go.microsoft.com/fwlink/?LinkId=108181" alt="Get
13. Microsoft Silverlight" style="border-style: none" /></a>
14. </object>
15.
```

```
<input type="text" id="MCText" disabled="disabled"
value="HTML Text Value from the browser." />
```

First, we need access to the browser, so at the top of our App.xaml.vb file, we need to import the browser namespace: **Imports System.Windows.Browser**

Next, we add code to Application_Startup as in the code below:

```
PrivateSub Application_Startup(ByVal o AsObject,
  ByVal e As StartupEventArgs) HandlesMe.Startup
  Me.RootVisual = New Page()

Dim doc As HtmlDocument = HtmlPage.Document
doc.GetElementById("MCText").
  SetProperty("disabled", False)
doc.GetElementById("MCText").
  SetAttribute("value", "Text from VB
Managed Code in Silverlight Object.")
EndSub
```

With the Imports System.Windows.Browser namespace imported, we have easy access to the browser's DOM object. We grab a reference to it from "HtmlPage.Document", and then we can use the SetProperty and SetAttribute methods to control the textbox in our web page. First, we enable the control by setting the disabled property to false, and then we populate the textbox with a string of our choosing by calling SetAttribute on the value attribute. Note, we have no JavaScript events or code that access our MCText textbox. It all happens from compiled code in our Silverlight Application.

Running the app shows that when the page first renders the textbox says "HTML Text Value from the browser", but in a second, when the Silverlight control is initialized the contents of the textbox change to read "Text from VB Managed Code in Silverlight Object.". Voila, Silverlight code is manipulating the DOM, but what if we want to call compiled code in the Silverlight application in response to some action that the user takes in the browser? We need to be able to call that Silverlight code from our client-side JavaScript. We can accomplish this by exposing a method in our Silverlight code as a "ScriptableMember." But first, we need a class in our Silverlight code that we want to expose to our client side JavaScript. So, I'll create a new Visual Basic Class named MyScriptableManagedType and decorate the class with the "ScriptableType" attribute. In the MyScriptableManagedType.vb file, I'll create a method to convert a string to all upper case. Then, we can "decorate" the method that we want to expose with the "ScriptableMember" attribute. Remember to import the System.Windows.Browser namespace. The code for our new class looks Listing 2.

Next, we need to register our new scriptable type with the web page, so in our Silverlight Application_Startup method, we need to add a couple of lines of code:

```
'Set upscriptable managed type for access from
  JavaScript.
Dim smt As MyScriptableManagedType = New
  MyScriptableManagedType()
HtmlPage.RegisterScriptableObject("mySLapp", smt)
```

The first line (after the comment) creates an object instance of our new class and the second line registers the object that we just created with the HTML DOM and tells the DOM to use the name "mySLapp" to reference it. That's all we need on the Silverlight side. On the JavaScript side, we need to implement an event handler for the Silverlight "onLoad" event, and in that handler, we need to fetch a JavaScript accessible reference to the loaded Silverlight Object. We can do this with a few lines of JavaScript code:

```
var slCtl = null;
// Create a JavaScript Variable to hold the
  Silverlight App Reference
function pluginLoaded(sender, args)
{
slCtl = sender.getHost();
// Get a reference to the Silverlight Application
// and assign it to the local JavaScript variable.
}
```

Before we can actually access our scriptable Silverlight object, we need to register our pluginLoaded method via the "onLoad" parameter to the Silverlight Application's object tag. We can just add a parameter line like this:
<param name="onLoad" value="pluginLoaded" />

Now, we can set up a textbox that needs our "CustomStringFormatter" method and a bit of JavaScript to call it. First, we'll add another textbox and a Button to our HTML, and we'll specify a Button Click Event Handler for the Button.

```
<input type="text" id="FormatedText"
  value="hello world." /> 
<input type="button" id="ButtonCustomFormatter"
  value="Format Text" onclick="return
  ButtonCustomFormatter_onclick()" />
```

Note, the value of the textbox is all lower case. All we need to do now is to implement the "ButtonCustomFormatter_onclick" event handler method in JavaScript.

```
function ButtonCustomFormatter_onclick()
{
 FormatedText.value = slCtl.Content.mySLapp.
```

```
    CustomStringFormatter(FormatedText.value);
}
```

Now, when we run the application and click on the Button, we see the text in the textbox change to upper case, but the code that actually changes the case of the string is compiled into our Silverlight application.

In the examples in Figure 6, we've called compiled code in the Silverlight application from some client-side JavaScript, and we've populated HTML DOM objects from compiled code in the Silverlight application, but what if we need to go the other way? We may also need to access properties in our Silverlight object from client-side JavaScript code, and we may need to call client JavaScript methods from compiled code in our Silverlight classes. Well, no problem. First, we'll set and get properties in our class that we named "MyScriptableManagedType". We need a property with accessors, so we'll add one to our scriptable type like this.

```
Dim _name AsString

<ScriptableMember()> _
PublicProperty Name() AsString
   Get
    Return _name
   EndGet
Set(ByVal value AsString)
     _name = value
EndSet
EndProperty
```

Note the "<ScriptableMember()>" attribute. For more information about attributes in Visual Basic, see http://msdn.microsoft.com/en-us/library/39967861(VS.80).aspx.

There is no executable logic implemented for this property, **Set** simply saves the value passed to it, and **Get** simply returns the current value of the property. Since the property is defined as a ScriptableMember, we can both "Set" and "Get" this property from client-side JavaScript code.

Let's add a Button whose event handler will set the Silverlight object property, another that will retrieve it, and a textbox to display the value.

```
<input type="button" id="ButtonSetBrowserType"
  value="Set Browser Type" onclick="return
  ButtonSetBrowserType_onclick()" />
<input type="text" id="TextBrowserType"
  value="" />
<input type="button" id="ButtonGetBrowserType"
  value="Get Browser Type" onclick=
  "return ButtonGetBrowserType_onclick()" />
```

Please take note of the button onclick event handlers which we will implement simply like Listing 3.

The last thing we need to do to close the Silverlight DOM interop loop is to call JavaScript functions from our compiled Silverlight code. First, we need a JavaScript function to call, so let's add one like this.

```
function globalJSMethod(strParamGGS)
  {
// Just pop up a window and show whatever is in the
// parameter.
  alert(strParamGGS);
  }
```

Then, we need Visual Basic code in our Silverlight application to call our JavaScript function. To do this, we'll add a Subroutine to the Visual Basic code in the App.xaml.vb file of our Silverlight Application. A Subroutine or "Sub" in Visual Basic is simply a Function that doesn't have a return value. See Listing 4.

So, we have a JavaScript function on the page to call and a Visual Basic method that calls it. All we need to do is wire up our user interface to invoke the Visual Basic Method. We'll need a page level event to catch so we'll set up an HTML button.

LISTING 2

```
Imports System.Windows.Browser

<ScriptableType()> _
PublicClass MyScriptableManagedType

<ScriptableMember()> _
PublicFunction CustomStringFormatter(ByVal str
  AsString) AsString
Return str.ToUpper
EndFunction

EndClass
```

LISTING 3

```
1. function ButtonSetBrowserType_onclick()
2. {
3. // Set the "Name" property in our Silverlight Object with
4. // the value of the browser application we're running in.
5.    slCtl.Content.mySLapp.Name = navigator.appName;
6. }
7.
8. function ButtonGetBrowserType_onclick()
9. {
10. // Fetch the "name" property value from the SIlverlight
11. // Object and display it in the textbox.
12.    TextBrowserType.value =slCtl.Content.mySLapp.Name;
13. }
```

LISTING 4

```
1. ' Execute Business Logic (Here, get the current time)
2. ' Call a global JavaScript method defined on the HTML page.
3. PrivateSub CallGlobalJSMethod(ByVal sender AsObject,
4.    ByVal e As EventArgs)
5. Dim strMS AsString = DateTime.Now.Millisecond.ToString()
6. Dim strTime AsString = "Silverlight Says " +
7.    ControlChars.NewLine + DateTime.Now.ToLongTimeString() +
8.    " MS = " + strMS
9. HtmlPage.Window.Invoke("globalJSMethod", strTime)
10. EndSub
```

```
<input type="button" id="btnCallJSMethod"
    value="Silverlight catches the DOM Event and
    Calls a JavaScript Method" />
```

Notice that our button definition does NOT contain a click event handler specification. It's just an HTML button sitting on the page. Our Silverlight application could catch any JavaScript event; we're just going to use a simple button click event. Now, we can configure our Silverlight Application to catch that button's click event, and then our Visual Basic / Silverlight code can call our JavaScript function named "globalJSMethod". To do this, we need to add two bits of code to your App.xaml.vb file. First, we need a delegate to the Visual Basic method that we want to bind to the HTML Button's click event. We create this by adding the following line of code to the Application_Startup event in the App.xaml.vb file:

```
' Create a delegate for local Visual Basic Method
Dim del AsNew EventHandler(AddressOfMe.
  CallGlobalJSMethod)
```

One last line of code in the App.xaml.vb file will connect the HTML Button's Click event to our VB code.

```
' Hookup that delegates up to the click event a
' button on the html page.
doc.GetElementById("btnCallJSMethod").
  AttachEvent("click", del)
```

Now, when we click on "btnCallJSMethod", the click event is caught in the Silverlight Application and the Visual Basic code is executed (see Figure 7).

Conclusion

The DOM integration capabilities of Silverlight matter to the PHP developer because it serves as the bridge between the applications that we write in PHP (or any dynamic web development language) and the new functionality for which we could use Silverlight. Of course, this is just the entry level. Now that you know how Silverlight code can interact with the web pages we generate from PHP, we can start to explore the kinds of things one can do inside Silverlight applications themselves. Silverlight is a vast technology with a plethora of features and capabilities.

For more information on Developing with Silverlight visit http://silverlight.net/learn/learnvideo.aspx?video=65683.

JOE STAGNER joined Microsoft in 2001 as a Technical Evangelist and is now a Microsoft.NET team Program Manager (Web Tools and Platform). A former Developer Community Champion with the Microsoft MSDN Team of Digital Blackbelt fame, his development experiences have afforded him the opportunity to create commercial software applications across a wide diversity of technical platforms from Mainframes, through UNIX and Linux, to Microsoft Technologies on the Intel and Mobile computing platforms. In recent years, Stagner has been focused on highly-performant, geoscalable web application architectures, multi-platform interoperability and–above all–writing secure code.

Configure and Optimize PHP on Windows

by Ruslan Yakushev and Hank Janssen

It has been pretty hard in the past to figure out how to install and configure PHP on Windows. Additionally, when you did get it installed, finding information on how to optimize PHP was even harder to find. Due to the recent work done by the PHP Community and Microsoft, doing exactly that is now easier than ever.

It has always been possible to run PHP on Windows using the Web server included with the operating system—Internet Information Services (IIS). However, many real-life PHP applications could not be hosted on Windows in a production environment due to limitations in the technologies IIS offered for running PHP applications, namely using the Common Gateway Interface (CGI) or using the PHP ISAPI extension.

CGI requires separate executables that are launched by the Web server for each request in order to process the request and generate dynamic responses. CGI enables reliable execution of PHP on IIS by executing exactly one request per process. However, PHP applications can have poor performance when running via CGI because of the high cost of starting and shutting down a Windows process for each request.

PHP applications that use the IIS high-performance, multi-threaded ISAPI interface have much better performance. However, they would often suffer from instability due to lack of thread safety in some popular PHP extensions.

To address these problems, built-in FastCGI support has been introduced in IIS 7.0, and a FastCGI extension

Related URLs

PHP: *Any*

Other Software:

- Windows XP SP2 or later (vista, 2000, server 2003, 2008 or win7)
- Microsoft IIS Web server

Useful/Related Links:

- PHP for Windows site http://windows.php.net
- PHP for Windows downloads http://windows.php.net/downloads
- Official IIS FastCGI Site: http://www.iis.net/fastcgi
- IIS Administration Pack: http://www.iis.net/extensions/AdministrationPack

Support Section:

- IRC Channel: Freenode #php-dev-win
- Windows Internals mailing list: http://www.php.net/mailing-lists.php

has been provided for IIS 5.1 and IIS 6.0. FastCGI addresses the performance issues inherent in CGI by providing a mechanism to reuse a single process for many requests. Additionally, FastCGI maintains compatibility with non-thread-safe libraries by providing a pool of reusable processes and ensuring that each process will handle only one request at a time.

Enabling FastCGI Support in IIS

The process for enabling FastCGI in IIS depends on what version of Windows you have. In Windows XP with IIS 5.1 and in Windows Server with IIS 6.0, FastCGI support is provided via a FastCGI extension which can be downloaded from the official IIS site. IIS 7.0 in Windows Vista Service Pack and in Windows Server 2008 has a built-in FastCGI module that you simply enable and configure.

Installing the FastCGI Extension for IIS 5.1 and 6.0

The FastCGI extension for Windows XP and Windows Server 2003 can be downloaded from the official IIS site (see Related URLs). When you run the FastCGI installer, it copies files for the FastCGI extension and then registers and enables the FastCGI Web server extension. At this point, the FastCGI support is enabled in IIS, but there is no FastCGI executable yet configured to work with IIS via FastCGI protocol. The section "Installing PHP" later in this document describes how to configure IIS to process PHP requests by using FastCGI.

FIGURE 1

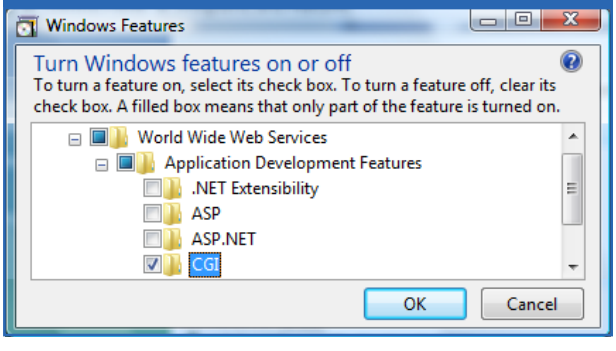
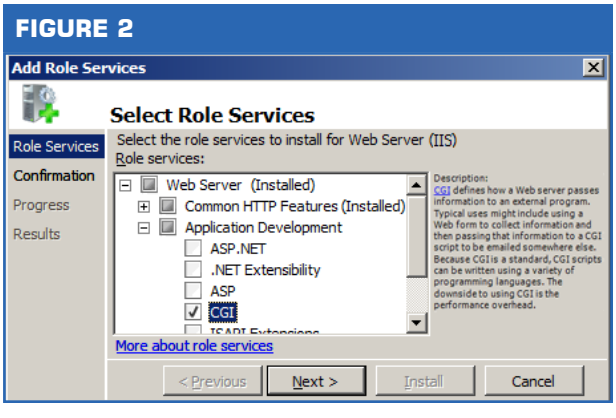


FIGURE 2



Enabling FastCGI in IIS 7.0

To enable FastCGI in IIS 7.0 on Windows Vista, the first thing you need to do is install SP1. SP1 provides FastCGI to IIS on Windows Vista. Add the CGI feature in the Windows Control Panel (*Control Panel* > *Programs and Features* > *Turn Windows features on or off*, as shown in Figure 1. This enables both the CGI and FastCGI services. On Windows Server 2008, add the CGI role service by using the Server Manager (*Server Manager* > *Roles* > *Add Role Services*), as shown in Figure 2. This enables both the CGI and FastCGI services.

You can configure settings for the FastCGI module in IIS 7.0 by using command-line tools or by manually editing the IIS configuration files. If you prefer to use a GUI for editing the FastCGI configuration, you can install IIS Administration Pack (see Related URLs).

For both Server 2008 and Windows Vista, you enable CGI. This in effect enables the FastCGI capability used by PHP. The Administration Pack provides a user interface for editing FastCGI settings in IIS Manager. See Figure 3.

After you enable support for FastCGI, it is time to install and configure PHP.

Installing PHP

Installing PHP on Windows is a straightforward process thanks to the PHP installer package available on the PHP site (see Related URLs). Be sure to download the installer for the latest non-thread-safe version of PHP, which at the time this article was written was PHP Version 5.2.9, Non-thread-safe installer. A non-thread safe build of PHP provides significant performance gains over the standard build by not doing any thread-safety checks. Thread-safety checks are not necessary, because FastCGI ensures a single-threaded execution environment.

After you have downloaded and launched the installer, accept the license agreement, choose the installation path, and then choose the installation option IIS

FastCGI. See Figure 4. Choosing this option tells the installer to perform the configuration steps in IIS that enable the server to handle PHP Web requests. For IIS 5.1 and IIS 6.0, the installer creates a script mapping that instructs IIS to pass all requests for PHP pages (.php resources) to the FastCGI extension. You can see this configuration by opening server properties and editing the extension mappings, as shown in Figure 5.

The FastCGI extension, in turn, is configured to pass the requests to the PHP executable that you installed earlier. You can see this configuration if you open the **fcgiext.ini** file, which is in the following location:
%WINDIR%\System32\inetsrv

The configuration file contains the following settings:

```
[Types]
php=PHP

[PHP]
ExePath=c:\php\php-cgi.exe
```

On IIS 7.0, the installer creates a handler mapping that, as with IIS 6.0, instructs the Web server to pass all PHP requests to FastCGI. You can examine this configuration by opening the Handler Mappings user interface in IIS Manager.

The installer also configures the FastCGI module to use the PHP executable for processing PHP requests. This configuration can be examined in the FastCGI settings in IIS Manager. See Figure 6.

FIGURE 3

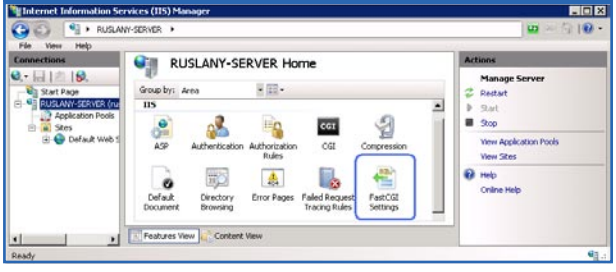
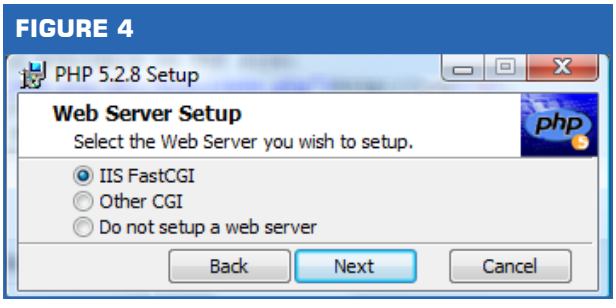


FIGURE 4



After the PHP installer completes the setup process, you can confirm that PHP is set up successfully. To do so, create a new text file named phpinfo.php in the following folder:
%SystemDrive%\inetpub\wwwroot

Open this file in a text editor, and add a line of PHP code that calls the phpinfo function. Save the file, and then request the following URL in a browser:
http://localhost/phpinfo.php

If PHP is working properly, you will see the standard PHP information page.

Configuring IIS and PHP

Now that you have IIS and PHP working together, it is time to perform a few configuration steps to ensure that you have an optimal execution environment for PHP applications.

Setting the Default Document

The first configuration task you want to do with PHP on IIS is to set the default document to **index.php**. To do that on IIS 5.1 and IIS 6.0, you use the IIS Manager tool. On IIS 7.0, you can use the IIS Manager UI. Alternatively on IIS 7.0, you can use the **appcmd.exe** command-line tool to set up **index.php** as the default document. Use the following syntax:

```
%WINDIR%\system32\inetsrv\appcmd.exe set config -
section:system.webServer/defaultDocument
/+"files.[value='index.php']" /commit:apphost
```

FIGURE 5

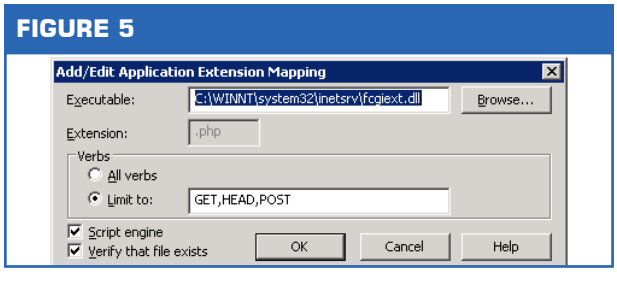
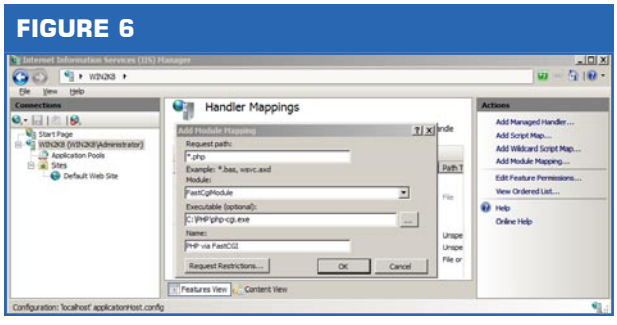


FIGURE 6



Optimizing PHP On Windows

Now that you have everything running, it is time to look at some optimizations you can do.

Firewall: To Disable or Not?

So, what steps can one take to optimize PHP performance on Windows? There is, of course, the Firewall, and turning that off will yield some improvements: we see a 0.2% improvement. This means that the Firewall is pretty efficient, and there is really no reason to turn it off. So, in a real world situation, leaving the Firewall enabled for security is definitely recommended.

Environment Variables

There are two environmental variables set; **InstanceMaxRequests** (Figure 7) and **PHP_FCGI_MAX_REQUESTS** (Figure 8) need to be set to a high value, "10000" in this case. Why is 10,000 a magic number? Indeed, the default number is 200, and performance under this default condition is horrible. However, increasing the number to 20,000 or 100,000 won't yield a better number either, so our test sticks to 10,000. Also, a **MaxInstances** value should be higher than the number of the system's CPU core. Theoretically, the number of cores should match the number of FCGI **MaxInstances**. However, after changing the number of **MaxInstances**, n cores equal to the **Maxinstances** does not show the optimal performance results. This is possibly related to context switching, so increasing this number slightly could result in better performance numbers. A drastic increase in this number negatively impacts performance. For example, for servers with 8 cores, our testing has shown that you get the best performance when FCGI **MaxInstances** number is set to "10".

FIGURE 7

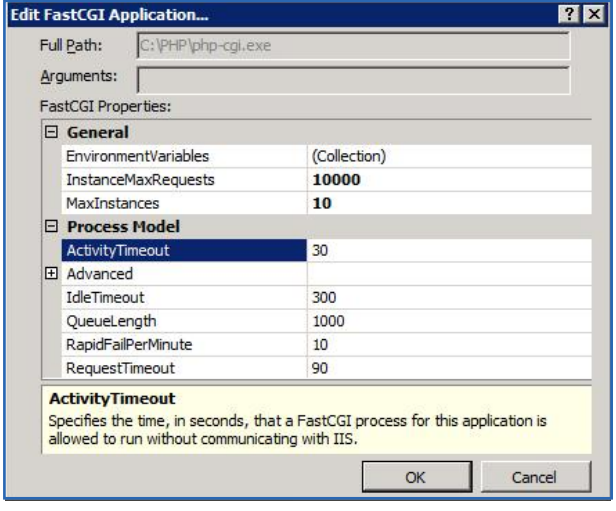
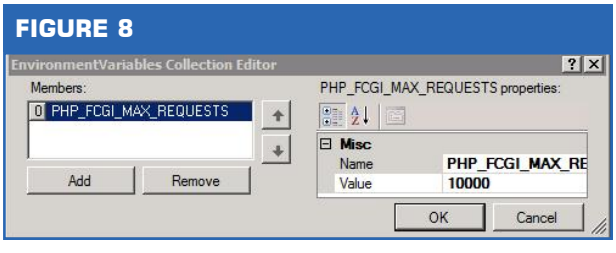


FIGURE 8



> How do you optimize PHP on Windows?

Recommended php.ini Settings

The following table lists settings that you can use to configure PHP for optimal performance and to make it more secure. To make the changes, locate and open the **php.ini** file, and then change the configuration settings to the values shown in Listing 1.

After you update the **php.ini** file, do not forget to recycle the IIS application pool that is used to host your PHP application. Recycling the application pool ensures that the PHP executable picks up the updated configuration settings. To quickly recycle an application pool in IIS 7.0, you can use the following command-line command:

```
WINDIR%\system32\inetsrv\appcmd recycle APPPOOL
"NameOfApplicationPool"
```

Disable luafv.sys

If you run PHP on Windows 2008 or Vista, there is a further step you can take to optimize performance. This step is most certainly optional and needs to be understood before you do this. This part certainly falls under the "User Beware" category, and you need to understand what it means to do this step and decide if it is the right thing to do for your particular system setup.

UAC comes with a driver called the UAC File Virtualization Filter Driver or **luafv.sys**. What does this

driver do? Well, this driver ensures that older installed legacy applications that want to write to protected areas (e.g: **windows\system32** etc) are intercepted and re-directed automatically to an area that the user is allowed to write to. This way, older applications that, for whatever reason, want to write to protected system directories do not fail with a write error. There is overhead with this, and, in some instances, we have seen that overhead reach 10% or so, even with applications that do not ever write to protected areas.

So, what happens if you disable this driver? Well, the short of it is that older legacy applications will fail when they want to write to an area they are not allowed to instead of being re-directed to an area they can write to. If you know your system does not have any of these applications installed, then you can disable this driver. This is what you need to do to disable it (WARNING: This means modifying the registry settings, as always, backup before you start!);

Run regedit (**Start > Run... > regedit**). To disable the driver, navigate to:

```
HKEY_LOCAL_MACHINE > SYSTEM > CurrentControlSet >
Services > luafv
```

Then, modify the variable **Start** to value **(Hex) 4**. If you want to re-enable this driver simply set the value to 2.

Running Multiple Versions of PHP Side-by-Side

Some PHP applications might rely on functions or features that are available only in certain versions of PHP. To host such applications on the same server, different PHP versions must run side-by-side. The IIS FastCGI handler fully supports running multiple versions of PHP on the same Web server.

Note that the PHP installer does not support installation of multiple PHP versions side-by-side. Therefore, it is recommended that you download the actual PHP binaries from the PHP site and then install and configure them using the procedures in the following sections. For example, assume that on your Web server you plan to support PHP 4.4.8, PHP 5.2.1, and the non-thread-safe version of PHP 5.2.5. In addition, you have three Web sites, and each site needs to use a different version of PHP. To enable this configuration, you must place PHP binaries in separate folders on the file system (for example, on **C:\php448'',** C:\php521", and **C:\php525nts**) and then configure FastCGI by following the configuration steps that are specific to the IIS version on your server.

Configuring IIS 5.1 and IIS 6.0 for Multiple Versions of PHP

To configure IIS 5.1 and IIS 6.0 for multiple versions of PHP, run the **fcgiconfig.js** script that is provided with the FastCGI extension. The script is located in the following folder: **%WINDIR%\system32\inetsrv**

The script will create FastCGI script mappings and

LISTING 1

Setting	Description
fastcgi.impersonate=1	FastCGI under IIS supports the ability to impersonate security tokens of the calling client. This setting allows IIS to define the security context that the request runs under.
cgi.fix_pathinfo=1	This provides PATH_INFO/PATH_TRANSLATED support for CGI. The previous behavior of the PHP CGI module was to set PATH_TRANSLATED to SCRIPT_FILENAME, and to ignore the PATH_INFO setting. For information about PATH_INFO, see the CGI specification. Changing this setting to 1 causes the PHP CGI to fix its paths to conform to the specification.
cgi.force_redirect=0	You must turn this off under IIS. Left undefined, PHP will turn this on by default. IIS takes care of CGI handling.
extension_dir	This setting specifies a location for PHP extensions. Typically for PHP 5.2.x, you use the following values for this setting: extension_dir="./ext" -or- extension_dir="C:\Program Files\PHP\ext"
allow_url_fopen=Off allow_url_include=Off	These settings disable remote URLs for file-handling functions. This can reduce code-injection vulnerabilities.
register_globals=Off	This setting disables register_globals, which prevents PHP from injecting your scripts with global variables that are defined based on Web request data.
open_basedir="c:\inetpub\"	This setting restricts where PHP processes can read and write in the server's file system.
safe_mode=Off safe_mode_gid=Off	These settings disable safe mode.
max_execution_time=30 max_input_time=60	These settings limit script execution time.
memory_limit=16M upload_max_filesize=2M post_max_size=8M max_input_nesting_levels=64	These settings limit memory usage and file sizes for PHP processes.
display_errors=Off log_errors=On error_log="path"	These settings configure error messages and logging.
fastcgi.logging=0	A PHP request to the IIS FastCGI module will fail if any data is sent on stderr by using the FastCGI protocol. Disabling FastCGI logging prevents PHP from sending error information over stderr, which prevents the Web server from sending HTTP 500 response codes to the client.
expose_php=Off	This setting hides the presence of PHP.

create process-pool definitions that use different versions of the PHP executables. Use the following syntax, making sure you substitute the correct site address and site ID for siteN.com and siteN_id respectively.

```
cscript fcgiconfig.js -add -section:"PHP siteN.com"
-extension:php - path:C:\php448\php.exe
-site:<siteN_id>
```

Configuring IIS 7.0 for Multiple Versions of PHP

To configure IIS 7.0 for multiple versions of PHP, use the IIS 7.0 **appcmd.exe** command line tool to create FastCGI process-pool definitions that use different versions of PHP executable. Use the following syntax, substituting the folder location where you installed the PHP binaries, making one entry for each of your three sites pointing to the version of PHP that it should be using:

```
C:\>%windir%\system32\inetsrv\appcmd set config
/section:system.webServer/fastCGI
/+[fullPath='c:\php448\php.exe']
```

Next, define handler mappings for each of those sites to reference a corresponding FastCGI application process pool. Note that each FastCGI process pool is uniquely identified by a combination of **fullPath** and **arguments** properties. Use the syntax in Listing 2, substituting the appropriate site name for your three Web sites.

Enabling Per-site PHP Configuration

When a single Web server is used to host multiple PHP Web sites, it might be necessary to have different PHP configurations for each Web site. With IIS FastCGI, you can allow each Web site to use its own version of the **php.ini** file. This section describes how to enable this configuration in IIS with FastCGI.

Per-Site Configuration in IIS 5.1 and IIS 6.0

Imagine that you have two Web sites in IIS 6.0 (website1.com and website2.com), and you want each Web site to use its own version of the **php.ini** file. Assume that the **php.ini** file for website1.com is located in the **C:\Inetpub\website1.com** folder and that the **php.ini** for website2.com is located in the **C:\Inetpub\website2.com** folder. The first step is to create script mappings and FastCGI configuration sections for each Web site. To do so, you can again use the fcgiconfig.js script, which is located in the following folder: **%WINDIR%\system32\inetsrv**

Run the following commands to create PHP script mappings for website1.com and website2.com. Make sure that you substitute the correct site address and site ID for websiteN.com and siteN_id, respectively. To find out the identifiers for each Web site, open IIS Manager, and select "Web Sites" node in a tree view on left-hand side. This will list the identifiers of all Web sites that are configured in IIS.

```
cscript fcgiconfig.js -add -section:"PHP
websiteN.com" -extension:php -
path:C:\php\php-cgi.exe -site:<siteN_id>
```

After you run these commands, open the **fcgiext.ini** file, which is located in the following folder: **%WINDIR%\system32\inetsrv**

The **fcgiext.ini** file will contain the following sections:

```
[Types]
; The site ids will be different for your site
php:169297538=PHP website1.com
php:273357939=PHP website2.com
site

[PHP website1.com]
ExePath=C:\php\php-cgi.exe

[PHP website2.com]
ExePath=C:\php\php-cgi.exe
```

The [PHP website1.com] and [PHP website2.com] sections can be used to specify site-specific FastCGI configuration settings. You can use these sections to specify the path to the **php.ini** file for each of the Web sites.

When the PHP process starts, it determines the location of the **php.ini** configuration file by using various settings. One of the places where the PHP process searches for the location of the **php.ini** file is the PHPRC environment variable. If the PHP process finds a **php.ini** file in the path specified in this environment variable, the process will use that path. Otherwise, the process reverts to looking in the default location for the **php.ini** file. You can use the following command, per site, to configure FastCGI to set this environment variable to point to the site-specific **php.ini** file, replacing websiteN.com with your information:

```
cscript fcgiconfig.js -set
-section:"PHP websiteN.com" -
EnvironmentVars:PHPRC:C:\Inetpub\websiteN.com
```

After you run these commands, when you examine the **fcgiext.ini** file, you will see the configuration sections updated as follows:

```
[PHP websiteN.com]
ExePath=C:\php\php-cgi.exe
EnvironmentVars=PHPRC:C:\Inetpub\websiteN.com
```

Per-Site Configuration in IIS 7.0

Creating an independent configuration for each PHP Web site in IIS 7.0 requires two procedures: creating a separate process pool for each site, and specifying the location of the **php.ini** file for each site.

Creating Per-site PHP Process Pools

If each Web site has its own application pool (which is a recommended practice for IIS 7.0), it is possible to associate a dedicated FastCGI process pool with each Web site. A FastCGI process pool is uniquely identified by the combination of fullPath and arguments properties when you use the **appcmd.exe** command to create the process pools. Therefore, if it is necessary to create several FastCGI process pools for the same process executable, such as **php-cgi.exe**, you can use the arguments attribute to distinguish process-pool definitions. In addition, with **php-cgi.exe** processes, the command line switch **-d** can be used to define an entry in the **php.ini** for the PHP process. This switch can be used to set a PHP setting that makes the arguments string unique. Assume that there are two Web sites (website1.com and website2.com), and each must have its own PHP settings. You can create the FastCGI process pools by using the following **appcmd.exe** commands:

```
appcmd.exe set config
-section:system.webServer/fastCgi
/+"[fullPath='C:\php\php-cgi.exe',arguments='-d
open_basedir=C:\inetpub\websiteN.com\']"
/commit:apphost
```

LISTING 2

```
C:\>%windir%\system32\inetsrv\appcmd set config site1 -
section:system.webServer/handlers
/+"..[name='PHP448_via_FastCGI',path='*.php',verb='*',
modules='FastCgiModule',
scriptProcessor='c:\php448\php.exe',resourceType='Either']

C:\>%windir%\system32\inetsrv\appcmd set config site2-
section:system.webServer/handlers
/+"..[name='PHP521_via_FastCGI',path='*.php',verb='*',
modules='FastCgiModule',
scriptProcessor='c:\php521\php-cgi.exe', resourceType='Either']

C:\>%windir%\system32\inetsrv\appcmd set config site3 -
section:system.webServer/handlers
/+"..[name='PHP525nts_via_FastCGI',path='*.php',verb='*',
modules='FastCgiModule',
scriptProcessor='c:\php525nts\php-cgi.exe', resourceType='Either']
```

User Group Directory

If your user group is not listed, send an e-mail to beth@phparch.com, and let us know the details so we can include it in our directory.

Sydney PHP Group (Sydney, Australia)
http://www.sydphp.org

PHPBelgium (Belgium)
http://phpbelgium.be

Ceara PHP Users Group (Ceara, Brazil)
http://cepug.org

PHPDF - Distrito Federal PHP Users Group (Brasília, DF, Brazil)
http://www.phpdf.org.br

GOPHP - Grupo de Desenvolvedores PHP de Goiás/Brazil (Goiás, Brazil)
http://www.gophp.com.br

PHP-Maranhão (São Luis, Maranhão, Brazil)
http://br.groups.yahoo.com/group/php-maranhao/

PHP MS - Users Group of Mato Grosso do Sul (Mato Grosso do Sul, Brazil)
http://www.phpms.org

PHP MG (Minas Gerais, Brazil)
http://www.phpmg.com

PHP-Paraïba (Paraïba, Brazil)
http://groups.google.com.br/group/php-pb/

PHPSC - Grupo de Usuários de PHP do Estado de Santa Catarina (Florianópolis, Santa Catarina, Brazil)
http://www.phpsc.com.br

PHP-SP (São Paulo, Brazil)
http://www.phpsp.org.br

PHP Rio (Rio de Janeiro, Brazil)
http://www.phprio.org

PHP London (London, England)
http://www.phplondon.org

Irish PHP Users' Group - Cork (Cork, Ireland)
http://www.php.ie

Irish PHP Users' Group - National (Dublin, Ireland)
http://www.php.ie

Irish PHP Users' Group (Monaghan, Ireland)
http://www.php.ie

New Zealand PHP Users Group (Auckland, New Zealand)
http://www.phpug.org.nz

Auckland PHP Meetup (Auckland, New Zealand)
http://php.meetup.com/10/

PHPBarcelona - PHP Barcelona User Group (Barcelona, Spain)
http://phpbarcelona.org

Front Range PHP Users Group (Colorado Springs, CO, USA)
http://www.frontrangephp.org

Atlanta PHP (Atlanta, GA, USA)
http://atlantaphp.org/

CIAPUG - Central Iowa Area PHP User Group (Des Moines, Iowa, USA)
http://www.ciapug.org

PDXPHP (Portland, OR, USA)
http://www.pdxphp.org

DallasPHP (Plano, TX, USA)
http://www.DallasPHP.org

In this example, the PHP setting **open_basedir** is used to distinguish between process-pool definitions. In addition, the setting enforces that the PHP executable for each process pool can perform file operations only within the root folder of the corresponding Web site. After creating the process pools, you can use the following command to create a handler mapping for website1.com and website2.com:

```
appcmd.exe set config "websiteN.com"
-section:system.webServer/handlers
/+"[name='PHP_via_FastCGI',
path='C:\php\php-cgi.exe|-d
open_basedir=C:\inetpub\websiteN.com\',
modules='FastCgiModule']"
```

Specifying the Location of the php.ini File

To tell the PHP process where to find the **php.ini** file, you can use the **PHPRC** environment variable. If the PHP process finds a **php.ini** file in the path specified in this environment variable, the process will use that path. Otherwise, the process reverts to looking in the default location for the **php.ini** file.

Imagine that you have two Web sites in IIS 7.0 (website1.com and website2.com), and you want each Web site to use its own version of the **php.ini** file. Assume that the **php.ini** file for website1.com is located in the **C:\Inetpub\website1.com** folder and that the **php.ini** for website2.com is located in the **C:\Inetpub\website2.com** folder. You can configure the FastCGI process pools to set the PHPRC server variable to point to the **php.ini** files located in root sites folders by using the following command for website1.com and website2.com:

```
appcmd.exe set config
-section:system.webServer/fastCgi
/+"[fullPath='C:\php\php-cgi.exe',arguments='-d
open_basedir=C:\inetpub\websiteN.com\'].
environmentVariables.[name='PHPRC',value=
'C:\inetpub\websiteN.com\']" /commit:apphost
```

This configuration also ensures that if there is no **php.ini** file found in the location specified by the PHPRC environment variable, PHP will fall back to using the default **php.ini** file located in the same folder where **php-cgi.exe** is located.

Even though PHP has gone through tremendous changes in the last 12 months and is constantly evolving, we hope this article has provided you with a good overview on how to configure and optimize PHP on Windows.

HANK JANSSEN is the Director of Program Management at the Microsoft Open Source Technology Center (OSTC) where he runs the Open Source Software Lab and Novell Joint Interoperability labs. Hank has been working with *nix and other OSS for over 20 years. He started his work at AT&T doing kernel programming for the SYS V process scheduler used for Digital Telephone switches (5ESS). At AT&T, he designed a globally distributed database system before people knew what they were, using such golden oldies as

Getting Started with the SQL Server Driver for PHP

by David Sceppa

Microsoft recently produced a PHP extension to communicate with SQL Server 2005 and 2008 databases, the SQL Server Driver for PHP. This article will cover the basic workings of this new extension, demonstrating how to connect to SQL Server, execute queries, retrieve results and manage transactions.

In July of 2008, Microsoft released version 1.0 of its first PHP extension - the Microsoft SQL Server Driver for PHP. This extension allows PHP developers to take advantage of recent key features of the Microsoft SQL Server engine from within their PHP applications. The SQL Server Driver for PHP supports all editions of both SQL Server 2005 and 2008, from Express to Enterprise, including support for all new SQL Server 2008 data types. All of the basic functionality expected of a database extension (connect, query, retrieve results, transactions) is supported by the extension, as well as features that developers working with SQL Server have come to expect, such as connection pooling, integrated authentication and encrypted connections.

Microsoft has made the source code for the SQL Server Driver for PHP publicly available on the Microsoft Codeplex site: http://www.codeplex.com/SQL2K5PHP.

Installing the SQL Server Driver for PHP

The SQL Server Driver for PHP is available on the Microsoft Download Center. The extension was primarily tested against version 5.2.6 of the PHP engine but should work against all 5.2.x versions starting with

Related URLs

PHP: *5.2.4+*

Other Software:

- Microsoft SQL Server 2005 or SQL Server 2008 (enable Full-Text Search service in order to install AdventureWorks examples)
- Microsoft SQL Server Driver for PHP http://go.microsoft.com/fwlink/?LinkId=123470
- Microsoft SQL Server Native Client ODBC Driver http://www.microsoft.com/downloads/details.aspx?FamilyId=50b97994-8453-4998-8226-fa42ec403d17&displaylang=en
- AdventureWorks Sample Database http://msftdbprodsamples.codeplex.com/Release/ProjectReleases.aspx?ReleaseId=18407

Supported Windows OS:

- Windows 2000 Service Pack 4
- Windows Server 2003 Service Pack 2
- Windows Server 2008
- Windows Small Business Server 2003
- Windows Vista Service Pack 1
- Windows XP Service Pack 2

5.2.4. We are working with the community to make sure the extension also works well in 5.3.x environments. To install the driver, download the driver package from the Microsoft Download Center (http://go.microsoft.com/fwlink/?LinkId=123470) and run the self-extracting .exe. The self-extracting .exe contains two .dll files: php_sqlsrv.dll (non-thread-safe) and php_sqlsrv_ts.dll (thread-safe). Place the appropriate .dll file in the PHP extension directory (such as **C:\PHP\ext**). Developers working with a non-thread-safe build of PHP (php5.dll) should use the non-thread-safe version of the driver while developers working with a thread-safe build of PHP (php5ts.dll) should use the thread-safe version of the driver. Note: Guidance on PHP on the IIS site (http://www.iis.net/php) recommends using the non-thread-safe version of PHP for performance and security reasons.

Once you've copied the appropriate build of the driver in the PHP extension directory, edit the php.ini file to tell the PHP engine to load the driver assembly. Developers working with the non-thread-safe version of the driver should add the following entry to the Dynamics Extension section: **extension=php_sqlsrv.dll**

Developers working with the thread-safe version of the driver should add the following instead: **extension=php_sqlsrv_ts.dll**

> The SQL Server Driver for PHP supports all editions of both SQL Server 2005 and 2008, from Express to Enterprise, including support for all new SQL Server 2008 data types.

Additional Required Components

The SQL Server Driver for PHP does not communicate directly with SQL Server. Instead, the driver relies on the Microsoft SQL Server Native Client ODBC Driver to handle the communication with SQL Server. The SQL Server Driver for PHP is essentially a very thin layer atop the ODBC driver. You can download the ODBC driver here: http://msdn.microsoft.com/en-us/library/cc296170(SQL.90).aspx.

Microsoft continues to provide an updated ODBC driver for each release of SQL Server, including service packs, which ensures that the SQL Server Driver for PHP will be able to provide support for future SQL Server releases without requiring major re-writes to the driver's code. This approach allows the SQL Server Driver for PHP to leverage existing features in the current ODBC driver (such as connection pooling) and easily make future SQL Server features (such as new data types) available to the PHP community in a timely fashion.

While the initial release of the SQL Server Driver for PHP is built on top of the ODBC driver written for SQL Server 2005, the SQL Server Driver for PHP supports SQL Server 2008 as well, including all new SQL Server 2008 data types.

Microsoft also recently demonstrated its continued commitment to ODBC through the introduction of a new version of ODBC (3.8) with the recent Windows 7 beta. For developers building new applications in C or C++, ODBC is still the most powerful, stable and best-performing technology for communicating with SQL Server.

In short, building the SQL Server Driver for PHP on top of the ODBC driver gives PHP developers a highly performant solution for SQL Server connectivity, for both current and future versions of SQL Server.

Connecting to Your SQL Server Database

Connecting to your SQL Server Database using the SQL Server Driver for PHP is easy. To open a connection, simply call the **sqlsrv_connect** function and specify the name of the desired server. An instance name (**ServerName\InstanceName**) can be supplied if you're trying to connect to a named instance of SQL Server. If you want to talk to an instance of SQL Server running on the same machine as your PHP code, you can specify either '(local)' or '.' as the server name. For example, the following line of code establishes a connection to the local machine.

```
$conn = sqlsrv_connect('.\SQLExpress');
```

The SQL Server Driver for PHP uses Windows Authentication by default. If you do not specify a user name or password, the driver will attempt to connect to the SQL Server instance using the credentials of the process that's running the code. Remember that code running in a Web-based PHP application will not run in a process using the user's credentials by default. For example, PHP code running in Microsoft Internet Information Services 7.0, the process will use the local IUSR account's credentials by default.

You can specify a number of different options, such as a user name, password, database, etc. when connecting to SQL Server by using the second argument in the **sqlsrv_connect** function. Supply the list of connection attributes and values in an array as shown here:

```
$connOptions = array('UID' => 'MyUser', 'PWD' =>
  'MyPassword', 'Database' => 'AdventureWorks');

$conn =
  sqlsrv_connect('.\SQLExpress', $connOptions);
```

The SQL Server Driver for PHP uses connection pooling by default. With this option enabled, the physical connection to the database is pooled rather than closed when you call **sqlsrv_close** or when the connection variable falls out of scope. When you attempt to connect to SQL Server, the driver first checks for any pooled connections that have matching location, database and credential information. If a matching pooled connection is found, that connection is re-used. Otherwise, a new connection is established. The first time a connection is used once it's been retrieved from the pool, the connection is reset prior to any activity on the connection. Resetting the connection ensures that temporary resources (such as temporary tables or stored procedures), or queries that alter the connection's context (such as changing databases using the USE statement), do not cause unintended effects.

For more information on connection options, including the ones mentioned earlier and other handy features (like encrypted connections), see the documentation for the **sqlsrv_connect** function: http://msdn.microsoft.com/en-us/library/cc296161(SQL.90).aspx

Checking for Errors

If a connection attempt fails, the **sqlsrv_connect** function returns **false**. You can then use the **sqlsrv_errors** function to retrieve error information on the last operation performed by the driver. The **sqlsrv_errors** function returns an array of arrays, where the inner array consists of name value pairs. You could use this code to check to see if the connection attempt failed, and display error information if a failure occurred:

```
if ($conn === false) {
    foreach(sqlsrv_errors() as $error) {
       echo 'SQLSTATE: '.$error['SQLSTATE']."\n";
         echo 'Code: '.$error['code']."\n";
         echo 'Message: '.$error['message']."\n";
    }
}
```

As a general rule, the driver's functions allow you to follow this error handling approach of checking to see if a call to the driver returned **false** and then using **sqlsrv_errors** to retrieve error information. Other coding examples omit error checking for the sake of brevity.

Executing a Query

The simplest way to execute a query using the SQL Server Driver for PHP is to call the **sqlsrv_query** function and specify both the connection on which you want to execute the query and the query string as shown here:

```
$sql = 'SELECT ReviewerName, EmailAddress
        FROM Production.ProductReview';
$stmt = sqlsrv_query($conn, $sql);
```

Fetching Results

Once you've executed a query, you can retrieve the results in a number of different ways. The simplest way is to call **sqlsrv_fetch_array**, which will return the contents of the current row as an array. By default, you can access the value for a particular field by specifying either the numeric index (0-based) or the field name. Once you've reached the end of the resultset, the **sqlsrv_fetch_array** function returns **null**.

LISTING 1

```
1. $sql = 'SELECT ReviewerName, EmailAddress, ReviewDate
2.         FROM Production.ProductReview
3.         WHERE ProductID = 937';
4. $stmt = sqlsrv_query($conn, $sql);
5.
6. while (sqlsrv_fetch($stmt)) {
7.     echo 'Name: '.sqlsrv_get_field($stmt, 0)."\n";
8.     echo 'Email: '.sqlsrv_get_field($stmt, 1)."\n";
9.     echo 'Date: '.sqlsrv_get_field($stmt, 2,
10.             SQLSRV_PHPTYPE_STRING(SQLSRV_ENC_CHAR))."\n";
11. }
```

```
while ($row = sqlsrv_fetch_array($stmt)) {
    echo $row['ReviewerName'].' - '.
      $row['EmailAddress']."\n";
}
```

You can also retrieve the contents of a row as a PHP object, rather than an array, by calling **sqlsrv_fetch_object**.

Fetch Results Field-By-Field

The SQL Server Driver for PHP also allows you to fetch the results of a query on a field-by-field basis using **sqlsrv_get_field**. As part of this functionality, you can specify the desired field type for each field you retrieve. For example, you may want to retrieve the contents of a SQL Server datetime column as a PHP string. The code in Listing 1 demonstrates some of the flexibility available via **sqlsrv_get_field**. The code calls **sqlsrv_fetch** to fetch the contents of each row, and then calls **sqlsrv_get_field** to retrieve the contents of each individual field. After reaching the end of the resultset, **sqlsrv_fetch** returns **null**, and we exit the while loop.

LISTING 2

```
$sql = 'SELECT ReviewerName, EmailAddress,
               ReviewDate, Comments
        FROM Production.ProductReview
        WHERE ProductID = 937';
$stmt = sqlsrv_query($conn, $sql);

while (sqlsrv_fetch($stmt)) {
    echo 'Name: '.sqlsrv_get_field($stmt, 0)."\n";
    echo 'Email: '.sqlsrv_get_field($stmt, 1)."\n";
    echo 'Date: '.sqlsrv_get_field($stmt, 2,
             SQLSRV_PHPTYPE_STRING(SQLSRV_ENC_CHAR))."\n";
    echo 'Comments:';
    $comments = sqlsrv_get_field( $stmt, 3,
             SQLSRV_PHPTYPE_STREAM(SQLSRV_ENC_CHAR));
    fpassthru($comments);
    echo "\n";
}
```

LISTING 3

```
function DisplayReviews($conn, $productId) {
    //Use the ? character as a parameter marker
    $sql = 'SELECT ReviewerName, EmailAddress,
                   ReviewDate, Comments
            FROM Production.ProductReview
            WHERE ProductID = ?';

    //Construct an array that contains the parameters
    $params = array($productId);

    //Execute the query and specify the parameters
    $stmt = sqlsrv_query($conn, $sql, $params);

    //Process the query results
    //...
}
```

Accessing the Contents of a Field as a PHP Stream

The **sqlsrv_get_field** function also allows you retrieve the contents of fields as PHP streams rather than as strings. We can change the previous query to return the contents of the Comments field as a PHP stream and then use the PHP `fpassthru` function to process the contents of the stream, as shown in Listing 2.

While this example displays the contents of a text-based field, the SQL Server Driver for PHP also supports retrieving binary fields as streams. If your application's images are stored in the database, relying on this functionality of the driver can greatly simplify your PHP code.

Working with Parameterized Queries

Although the SQL Server Driver for PHP allows you to execute any Transact-SQL string, Microsoft does not recommend concatenating user-input with query strings. Following this approach can lead to SQL injection, potentially allowing the user to change the query in ways that the developer may not have expected and possibly leading to security problems within the application.

The recommended approach for handling user input as part of the query is to use a parameterized query. For example, the previous query searched for product reviews where the product reviewed had a product ID of 937. Imagine that the code is part of a function called DisplayReviews where the desired product ID is specified at run-time.

Executing parameterized queries using the SQL Server Driver for PHP is simple. In the code that follows, we replace the original value of 937 in the query with the parameter marker character ?. Then, we construct an array that contains the parameter values and pass that array into the **sqlsrv_query** function as shown in Listing 3.

Calling Stored Procedures with Parameters

In the previous example, the SQL Server Driver for PHP inferred a lot of information about the parameter used in the query. The driver determined that the PHP type was an integer and assumed that the SQL Server data type was an integer as well. The driver also assumed that the parameter was an input parameter.

For simple queries, these assumptions will work well,

but for complex queries you may need to specify the PHP or SQL Server data types and/or the parameter direction. For example, you may want to call a stored procedure that uses output parameters to return information.

The SQL Server Driver for PHP lets you specify this additional information by passing an array of arrays to the **sqlsrv_query** function. The entries in the outer array represent individual parameters. The entries in the inner arrays represent the value, the direction, the PHP data type and the SQL Server data type, respectively. You can omit individual entries and SQL Server will infer the missing piece(s) of information for the parameter(s).

> The recommended approach for handling user input as part of the query is to use a parameterized query.

For example, Listing 4 creates a simple stored procedure that adds two integers and returns the sum via an output parameter. The code then demonstrates how to call that function and specify an output parameter. The query string uses the canonical CALL syntax where CALL precedes the stored procedure name, parameter markers are enclosed in parenthesis and the entire query string is enclosed in curly braces.

Using Prepared Queries

In all of the query examples we've examined so far, we've called the **sqlsrv_query** function. The SQL Server Driver for PHP also supports the commonly used prepare / execute model, which can improve performance in scenarios where you want to execute the same query multiple times and change only the parameter values used. You can call **sqlsrv_prepare** using the same arguments as **sqlsrv_query** and the SQL Server Driver for PHP will prepare the query for execution. You can later call **sqlsrv_execute** to execute the query. You can also modify the contents of your parameter variables between calls to **sqlsrv_execute** and the SQL Server Driver for PHP will pass the new values to the database each time. Listing 5 demonstrates this approach.

Updating Data

Apart from the stored procedure example, each of the queries we've executed have been simple SELECT queries. In many applications, you need to allow the user to modify the contents of the database. This is generally achieved by using INSERT, UPDATE and DELETE queries, or stored procedures that execute such queries.

The SQL Server Driver for PHP does not parse the query string supplied and has no knowledge of whether the query is a SELECT query that returns rows or an INSERT, UPDATE or DELETE query that is expected to modify the contents of the database. It merely passes the query string and any parameter information to the database and retrieves the reported results.

Most developers use optimistic concurrency checks for UPDATE and DELETE queries to make sure the attempt to update the desired row only succeeds if the contents of the row have not been changed. If the contents of the desired row have been changed, the UPDATE or DELETE query will not modify any rows. Neither the SQL Server database engine nor the SQL

LISTING 4

```
$sql = 'CREATE PROCEDURE MySum
        (@input1 int, @input2 int, @output int OUTPUT)
        AS
        SET @output = @input1 + @input2';
$stmt = sqlsrv_query($conn, $sql);

$sql = '{call MySum(?, ?, ?)}';
$input1 = 21;
$input2 = 34;
$output = 0;  //Sets the PHP type for the output parameter
$params = array( array($input1, SQLSRV_PARAM_IN),
                 array($input2, SQLSRV_PARAM_IN),
                 array($output, SQLSRV_PARAM_OUT));
$stmt = sqlsrv_query($conn, $sql, $params);

echo $input1.' + '.$input2.' = '.$output."\n";
```

LISTING 5

```
$sql = 'SELECT ReviewerName, EmailAddress
        FROM Production.ProductReview
        WHERE ProductID = ?';
$productId = 0;
$stmt = sqlsrv_prepare($conn, $sql, array($productId));

$productIds = array(937, 709, 798);
foreach ($productIds as $val) {
    $productId = $val;
    sqlsrv_execute($stmt);
    echo 'Reviewers for ProductID: '.$productId."\n";
    while ($row = sqlsrv_fetch_array($stmt)) {
        echo '  '.$row['ReviewerName'].
             ' - '.$row['EmailAddress']."\n";
    }
    echo "\n";
}
```

Server Driver for PHP will report an error since the query ran successfully. To determine if the query affected the desired number of rows, use the **sqlsrv_rows_affected** function:

```
$sql = 'UPDATE Production.Product
    SET ListPrice = ListPrice * 2
    WHERE ProductID = ? AND ListPrice = ?';

$productId = 999;
$listPrice = 539.99;
$params = array($productId, $listPrice);

$stmt = sqlsrv_query($conn, $sql, $params);

echo sqlsrv_rows_affected($stmt)." rows affected";
```

Running this code once should double the list price for the specified product, assuming the list price for the product is initially set to $539.99 and the **sqlsrv_rows_affected** function will indicate that one row was affected. Running the query a second time will not affect any rows because the list price for the specified product is no longer $539.99 and the **sqlsrv_rows_affected** function will indicate that no rows were affected by the query.

LISTING 6

```
1. sqlsrv_begin_transaction($conn);
2.
3. $sql = 'UPDATE Production.Product
4.         SET ListPrice = ListPrice * 2
5.         WHERE ProductID = ? AND ListPrice = ?';
6. $productId = 999;
7. $listPrice = 539.99;
8. $params = array($productId, $listPrice);
9. $stmt = sqlsrv_query($conn, $sql, $params);
10.
11. $rowsAffected = sqlsrv_rows_affected($stmt);
12. if ($rowsAffected === 1) {
13.     sqlsrv_commit($conn);
14.     echo "Transaction committed\n";
15. } else {
16.     sqlsrv_rollback($conn);
17.     echo 'Query affected '.$rowsAffected." row(s)\n";
18.     echo "Transaction rolled back\n";
19. }
```

LISTING 7

```
1. $sql = 'INSERT INTO Orders (CustomerID, OrderDate)
2.                 VALUES (?, ?);
3.         SELECT SCOPE_IDENTITY() AS OrderID';
4. $customerId = 'ALFKI';
5. $orderDate = date_create();
6. $params = array($customerId, $orderDate);
7.
8. $stmt = sqlsrv_query($conn, $sql, $params);
9. echo sqlsrv_rows_affected($stmt)." row(s) affected\n";
10.
11. sqlsrv_next_result($stmt);
12. $row = sqlsrv_fetch_array($stmt);
13. echo 'New OrderID: '.$row[0]."\n";
```

Working with Transactions

The SQL Server Driver for PHP supports executing queries within transactions. Let's say that in the previous example, we want to make sure that the changes made by the UPDATE query are committed only if the query affected one row.

To begin a transaction, call the **sqlsrv_begin_transaction** function. Then, simply execute your queries. All work done on the connection will be handled within the transaction. You can later decide to commit the work by calling **sqlsrv_commit**, or roll back the work by calling **sqlsrv_rollback**. Listing 6 demonstrates this approach. Once you've called **sqlsrv_commit** or **sqlsrv_rollback**, there is no longer an active transaction on the connection and changes made on the connection will once again auto-commit.

Avoid long-running transactions by either committing or rolling back the transaction as quickly as possible. If a connection has an active transaction at the time it is closed (either directly or through garbage collection) the transaction will be rolled back. If connection pooling is enabled, the transaction will not be rolled back until the connection is re-used or is physically closed.

Retrieving New Auto-Increment Values

When inserting data, many developers let the database generate new primary key values using auto-increment columns. After executing an INSERT query that generates a new primary key value using an auto-increment column, you'll likely want that new key value. In SQL Server, the easiest way to return this information is to use the SCOPE_IDENTITY function, which returns the last auto-increment value inserted within the same scope - i.e. the same stored procedure, trigger, or batch.

The code in Listing 7 inserts a new row in an Orders table that uses an auto-increment column for its primary key, and then uses the SCOPE_IDENTITY function to retrieve the server-generated auto-increment value. Note the call to **sqlsrv_next_result**, which we'll discuss next, prior to fetching the newly generated auto-increment value.

For more information on the SCOPE_IDENTITY function, see the entry in SQL Server Books Online: http://msdn.microsoft.com/en-us/library/ms190315.aspx

Handling Multiple Results

SQL Server allows you to execute multiple queries in a batch. The **sqlsrv_next_result** function allows you to move to the next result within the batch. In the previous batch query, the initial result contains no columns and no rows. The result of the INSERT query is simply a message that indicates that the query affected one row, the row inserted by the query. In order to retrieve the results of the SELECT query, we need to call **sqlsrv_next_result** prior to fetching the new auto-increment value.

If you're processing the results of a batch query, the **sqlsrv_next_result** function will return **true** as long as there's another result to process. When there are no more results to process, **sqlsrv_next_result** returns **null**.

Developers who want to suppress the messages that indicate how many rows were affected can use SQL Server's NOCOUNT setting, described in more detail in the SQL Server documentation: http://msdn.microsoft.com/en-us/library/ms189837.aspx

Cleaning Up

If a SQL Server Driver for PHP's connection or statement falls out of scope, it will be cleaned up as part of the PHP engine's garbage collection process, and the resources associated with the connection or statement will be released at that time. Another option is to proactively release those resources. You can free statements using **sqlsrv_free_stmt** and close connections using **sqlsrv_close**.

What's In Store For the Next Version of the Extension?

The SQL Server Driver for PHP team is currently working on the next version of the extension. The feature request we've received most frequently via blogs, forums, etc. is to add UTF-8 support to the extension. We're in the process of adding that functionality, along with support for scrollable resultsets and row count. Naturally, we'll also make sure the driver runs well on PHP 5.3.

To keep up with the latest announcements about the extension, you can access the SQL Server Driver for PHP team's blog at: http://blogs.msdn.com/sqlphp

Is the Documentation Available Online?

The download for the SQL Server Driver for PHP includes a .chm file so you can access the driver's documentation locally, but you can also access the driver's documentation online at: http://go.microsoft.com/fwlink?linkid=119889

Where Do I Go to Ask Questions?

Members of the SQL Server Driver for PHP team and many members of the PHP community answer questions on the MSDN forum for the extension. We welcome your feedback and questions on how to use the current version of the extension as well as your input on how we can improve on it going forward. PHP developers who want to help answer other developers' questions are also welcome. Our forum can be found at the following URL: http://social.msdn.microsoft.com/Forums/en-US/sqldriverforphp/threads/

DAVID SCEPPA is the Program Manager for the Microsoft SQL Server Driver for PHP. He has worked at Microsoft for over thirteen years, focusing primarily on data access programming and is also the author of multiple books on Microsoft data access technologies.

PRE-LAUNCH CHECKLIST

by Eric David Benari

Shortly after most new sites are launched, the creators discover that some necessary elements have been forgotten. Pilots employ a pre-flight checklist to avoid problems while airborne, so it is only natural that web sites should have a pre-launch checklist to avoid problems once users begin visiting the site. This is more of a detailed checklist than a tutorial, but every topic in the list can easily be Googled for more info if needed.

Related URLs

PHP: *5.0.2+*

Other Software:

- Apache with headers, expires and deflate modules enabled

Related URLs:

- Validate your code: http://validator.w3.org
- Minification: http://code.google.com/p/minify
- Google Webmaster Tools: http://www.google.com/webmasters/tools
- YSlow: http://developer.yahoo.com/yslow
- memcached: http://www.danga.com/memcached

Error Management

Does your site have a *404 Not Found* page that includes links to functional pages?

If not, you may lose a potential visitor who follows an old link to a page that is no longer on your site. Ideally, this should be a *smart-404* page that attempts to provide suggestions as to what the user might have been looking for. At the very least, stick a Google site-search box on there, pre-populated with the name of the page that was not found (See Listing 1). Applying the same treatment to the *403 Forbidden* and other error pages is a good practice as well.

Valid Markup

Does every page validate as XHTML (or at least HTML)?

If not, your site is not likely to display consistently between browsers and will probably be a real mess when browsing from a mobile phone.

Basic Accessibility

Does your site render consistently between IE 6, IE 7, IE 8, FireFox 3, Safari and Google Chrome? Does your site display correctly on both Windows and OSX? Is the site usable at different screen resolutions? When you increase the default browser font-size, is the site still usable? Is the site legible with CSS disabled? Is it functional with JavaScript disabled?

If you have answered no to any of the above, then a percentage of your users will receive an inferior user experience.

LISTING 1

```
<style type="text/css">
/*<![CDATA[*/
<!--
#goog-wm {}
#goog-wm h3.closest-match {}
#goog-wm h3.closest-match a {}
#goog-wm h3.other-things {}
#goog-wm ul li {}
#goog-wm li.search-goog {display: block;}
//-->
/*]]>*/
</style>
<script type="text/javascript">
//<![CDATA[
<!--
var GOOG_FIXURL_LANG = 'en';
var GOOG_FIXURL_SITE = 'http://www.YOURSITE.com/';
//-->
//]]>
</script>
<script type="text/javascript" src="http://linkhelp.clients.google.com/tbproxy/lh/wm/fixurl.js"></script>

```

Minimal Semantics

Does every page have a unique and relevant title, meta-description and meta-keywords?

```
<title>Pre-Launch Checklist by Eric David Benari</
title>
<meta name="description" content="Shortly after most
new sites are launched the creators discover that
some necessary elements have been forgotten; this
can easily be avoided by following the steps in this
pre-launch checklist." />
<meta name="keywords" content="RAM disk, minify,
gzip, semantics, accessibility, favicon,
AllowOverride, caching" />
```

Without this, your search engine rank will suffer. Relevant titles also improve the user experience when someone bookmarks your site or looks at the browsers' history.

Client-Side Caching

Have you explicitly defined **expires** headers and disabled **FileETag** where appropriate?

This can drastically reduce the number of requests on your web server. Apache users can do this easily via the config files, either in **.htaccess** or **httpd.conf** (See Listing 2).

LISTING 2

```
<Directory "r:\YOURSITE">
    <IfModule mod_deflate.c>
        SetOutputFilter DEFLATE
        BrowserMatch ^Mozilla/4 gzip-only-text/html
        BrowserMatch ^Mozilla/4\.0[678] no-gzip
        BrowserMatch \bMSI[E] !no-gzip !gzip-only-text/html
        SetEnvIfNoCase Request_URI \
        \.(?:gif|jpe?g|png)$ no-gzip dont-vary
        <IfModule mod_headers.c>
            Header append Vary User-Agent env=!dont-vary
            Header unset ETag
        </IfModule>
    </IfModule>
    FileETag None

    Options FollowSymLinks
    Allow from all

    AllowOverride None
    Include "r:\YOURSITE\.htaccess"
</Directory>

<IfModule mod_expires.c>
    <Directory "r:\YOURSITE\css">
        ExpiresActive on
        ExpiresDefault "access plus 1 month"
    </Directory>
    <Directory "r:\YOURSITE\images">
        ExpiresActive on
        ExpiresDefault "access plus 1 month"
    </Directory>
    <Directory "r:\YOURSITE">
        <Files favicon.ico>
            ExpiresActive on
            ExpiresDefault "access plus 1 month"
        </Files>
    </Directory>
</IfModule>

```

Control Search Engine Indexing

Do you have a **robots.txt** in your webroot?

All web sites should have a valid **robots.txt** file even if every page on the site is meant to be indexed:

```
User-agent: *
Disallow:
```

Neglecting to have the file will bloat your web server error logs with *404 'robots.txt' not found* messages from all the search crawlers trying to read the non-existent file.

Favicon

Does your site have a **favicon** file?

The icon itself is of questionable value considering that it is rarely even noticed, but if you omit the **favicon** then each visitor to your site will add an additional *404 'favicon.ico' not found* error message to your log files. This is similar to the previous **robots.txt** issue but considerably more severe; only bots and search crawlers request the **robots.txt**, but the browser of every visitor coming to your site requests the **favicon**. If you are logging to the same hard disk that serves the web site (not an ideal practice), then this will also cause a minor performance penalty as the disk needs to write to the log file for each new visitor in between serving the files on your web site.

LISTING 3

```
<?php
$urls = array('/' => 'daily',
              '/news' => 'hourly',
              '/contact' => 'monthly');

echo '<?xml version="1.0" encoding="UTF-8"?>' . PHP_EOL
   . '<urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">' .
PHP_EOL;

foreach ($urls as $loc => $changefreq) {
    echo "\t<url>" . PHP_EOL
       . "\t\t<loc>http://www.YOURSITE.com" . $loc . '</loc>' . PHP_EOL
       . "\t\t<changefreq>" . $changefreq . '</changefreq>' . PHP_EOL
       . "\t</url>" . PHP_EOL;
}

echo '</urlset>';
```

LISTING 4

```
<?xml version="1.0" encoding="UTF-8"?>
<urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">
    <url>
        <loc>http://www.YOURSITE.com/</loc>
        <changefreq>daily</changefreq>
    </url>
    <url>
        <loc>http://www.YOURSITE.com/news</loc>
        <changefreq>hourly</changefreq>
    </url>
    <url>
        <loc>http://www.YOURSITE.com/contact</loc>
        <changefreq>monthly</changefreq>
    </url>
</urlset>
```

Minify JS and CSS files

Are your external files as small as they can get?

Minification is the act of stripping out whitespace and comments as well as implementing other space saving techniques such as consolidating CSS statements and refactoring JavaScript variables to minimize size (e.g. variables **showAdvancedSearch** and **isModernBrowser** might become **v1** and **v2**). This typically reduces the file size by about 15%, although this amount varies depending on the minification scheme as well as your commenting and white-space habits. It is critical to always verify that your minified JavaScript still works in IE; the JavaScript engine in IE requires a space after certain statement types that FireFox is able to execute flawlessly with one character following the next.

Consolidate JavaScript and CSS

Are you avoiding inline JavaScript and CSS? Are you serving just one JS and one CSS file?

Moving your inline JavaScript and CSS into external files allows the browser to cache the content. Consolidating them all into one JS and one CSS file also reduces the number of requests needed to load the page. This is the best-practice way of doing things, but in the real world it is not always the optimal way, especially if many of your pages have JS or CSS that is solely used on one page of your web site.

Placing JS and CSS inline can often yield a faster page load if adding a few bytes of un-cached inline code will reduce an HTTP request or considerably shrink the size of the consolidated global file.

Search Submission

Do you have a valid XML sitemap? Have you submitted your site for search engine indexing?

Register your domain in Google Webmaster Tools, create the validation file as instructed by Google and then create an XML sitemap as described in the *Sitemaps* section of Webmaster Tools. Listing 3 shows an example of how to easily generate a sitemap in PHP. The XML that gets output can be seen in listing 4. After launch, you will need to log into Webmaster Tools and tell Google the URL of the sitemap for crawling. This will

get Google to queue indexing of your new site so that it will show up in the Google search results.

Next, add your sitemap to your robots.txt:

```
User-agent: *
Disallow:
Sitemap: /sitemap
```

Then submit the site to Yahoo! Directory, Yahoo! Search, MSN and DMOZ. To conquer the hundreds of other search engines, I find it easiest to use a URL-submission service which submits the site to many engines with a few quick clicks.

> Serve all your static files from a RAM disk.

Trim Down Headers

Are your server responses only consisting of valuable data?

Turn off **expose_php** in your **php.ini** file, and remove any unnecessary page headers to reduce the size of each request. Also, check the site with *YSlow* and follow its suggestions.

Compress Throughput

Have you enabled **gzip** compression on your web server?

Gzip greatly improves your network throughput at the cost of a negligible amount of CPU overhead. Listing 2 contains an example of enabling gzip within the Apache configuration file.

Domain Distribution

Are the external files accessed by your site (images, JS, CSS, etc.) distributed over four domains?

This may seem strange, but four is a magic-number! Many browsers, by default, will not make more than two simultaneous connections to a single domain, but it will make many connections to different domains simultaneously. If all your content is served from a single domain, then the browser backs up into a queue as it requests two files at a time. Spreading your files over many domains (and you can use sub-domains like images.yourdomain.com) resolves the issue but at the cost of a slight delay as each domain needs to be resolved before any requests can be sent to it. In most situations, four sources of data is the ideal balance between simultaneous request limitations and domain resolution latency.

RAM

Are you serving every static element of your site from your server's RAM?

The most-effective way to maximize your server's traffic-handling capabilities is to avoid disk IO as much as possible. Serve all your static files from a RAM disk. Since RAM data will be erased if the server reboots or the OS crashes, you will need to create a shell/batch script that automatically restores the files from the hard disk upon reboot.

The following Windows DOS commands will restore files to the RAM disk **R:** from a HDD-based location on the **C:** drive.

```
mkdir R:\your_site
xcopy C:\your_site R:\your_site /E/H
```

Now, simply put the above code into a text-file with a **.bat** extension, and then create a shortcut to the file in your Windows startup folder. The folder location varies with Windows versions, but the path is not really needed, you can just double-click the **Startup** folder in the Start Menu to open the folder.

MySQL users can also create tables using the **memory** engine which stores the data in RAM. This is ideal for data that rarely changes such as a lookup table that indexes country codes. Just like with the RAM disk, the data in a **memory** table will be lost upon reboot so you need to create a SQL script to repopulate your **memory** tables from disk-based tables.

```
insert into countries_memory select * from
countries_innodb;
```

You can then set the file to execute automatically on reboot by adding the **init-file** directive to your **my.ini** file (or **my.cnf** on *nix). While you are editing the MySQL configuration, you should also be sure that query cache is enabled and that the cache memory limits are set appropriately for your hardware (the default settings were created many years ago, and do not take advantage of modern computing power).

For all dynamic data that is suitable for caching (both file-based data and database queries), use

memcached or another caching package. With **memcached**, you gain the additional benefit of being able to store your PHP session data in **memcached** as opposed to in files (the default) or in the database (an option best avoided).

Preload Configuration Files

Are you preloading configuration files?

Apache users should preload all **.htaccess** files and then disable on-the-fly **.htaccess** parsing. This saves a whole lot of file IO on your web server but also requires you to restart your web server whenever you want to implement a change made to an **.htaccess** file. In the section of your **httpd.conf** file that defines your site, remove any existing **AllowOverride** statements and add the following:

```
AllowOverride None
Include "/YOURSITE/.htaccess"
```

Restart the web server for the change to take effect. A complete example can be seen in Listing 2.

Error Handling

Are PHP error messages being logged but not displayed on the page? Is error reporting set at **E_ALL** or higher level?

Be sure that in the **php.ini** file of your production server, **log_errors** is on, **display_errors** is off and `error_reporting` is set to **E_ALL** or higher (ideally **E_ALL | E_STRICT**). This is especially important if your development and production environments are running different versions of PHP or if there are any server-related software configurations that are not identical between environments.

Security

Can passwords or back-end source code be accessed by simply navigating to a URL?

Some common examples to check are:

- http://YOURSITE.com/.svn
- http://YOURSITE.com/.htpasswd
- http://YOURSITE.com/config.inc

SSL (If Needed)

Did you install the SSL certificate? Can you access your site via both **http://** and **https://** without receiving any invalid-certificate alerts or similar warnings?

If your site will need to transmit to the browser securely over the HTTPS protocol, then you will need to install an SSL certificate that is registered to your site's domain name and issued by a real certificate authority (not self-signed). If you access your site over SSL by any means other than the domain name (e.g. IP address), then the browser will warn you about an invalid certificate. If your domain does not yet point to the server, then you will need to confirm that this warning will not occur post-launch. The way to do this is to open your operating system's **hosts** file in a text-editor (on Windows this is typically found at **C:\Windows\System32\drivers\etc\hosts**), and add the following to it on a new line:

```
127.0.0.1 YOURSITE.com
```

The hosts file is usually loaded and cached upon browser startup so you will need to close any browser windows that are already open and then open a new browser window. Now, navigate directly to your web site via the domain name (**http://YOURSITE.com**), and the page should load normally. Next, verify SSL (**https://YOURSITE.com**), and you should not receive any browser warnings.

Email

If this is a new domain, have you set up the necessary email accounts?

Verify that every email address appearing on your site is functional.

Suggestions?

If your pre-launch routine involves any other elements, then please contact me through http://www.ericdavid.cc, and send them my way!

ERIC DAVID BENARI is chairman of the IT Management Conference (www.ManageIT.me), organizer of the NY Enterprise 2.0 meetup and the sixth person in NY to become a Zend Certified Engineer (of PHP). He is also a PMI-Certified Project Management Professional, MySQL 5.0 Certified Developer and Certified DBA. He has been invited to speak at New York University (NYU), Yeshiva University, Stern College, as well as various IT conferences and has written for phparchitect on security, e-commerce, accessibility and IT best-practices.

The Cost of Security

by Arne Blankerts

When asked, almost everyone agrees there is need for better security in (PHP-)software: Maybe by designing the software with security in mind and thus writing a secure application or by frequently auditing existing code. Yet hardly anyone is willing to spend time, and therefore money, on it: It doesn't provide you with the latest cool feature or look fancy. It seems you don't gain anything except maybe the respect of other developers. It's like writing documentation: It's the most boring thing to do when you could be hacking the next cool thing(tm) instead. From a marketing perspective, simply claiming your website, application or library is secure until proven otherwise is way cheaper! In Non-OpenSource software, nobody but you knows the code and its potential problems anyway. So you're safe. Aren't you?

Considering the money lost either directly due to fraud and abuse or indirectly by time lost recovering a hacked system or software, the "let's hope for the best"-approach is of arguable quality. Spending time on a tiny security problem that, when exploited, will not cause any "real" trouble or bad press might be considered a waste of money. The key factor is to find a way to judge where, when and to what level security needs to be focused on.

Obviously, there's no such thing as 100% when talking about security, so it's vital to define the risks and results a failure in any of the named categories may cause. For example, let's start with the server the application is running on: The housing location and type of uplink are as important as the operating system and software, the versions in use and the hardware platform.

Hardening the system (removing unneeded services that come with a default installation or enforcing security rules like limiting shell access to a certain set of IP addresses and users) should be considered part of the installation process. It is not a reoccurring task per se, yet it's an important prerequisite for a secure environment. Thinking about the uplink and location for the server might not look very relevant on first glimpse but it is yet another key factor in the broader picture: No matter how much time and money you spend on securing the services reachable via remote connections, it's wasted if there is no security protocol for physical access to the machine. There's also no point in securing only one system while leaving other parts of the infrastructure unprotected: Imagine a webserver protected by an application level firewall hosted on a sealed box making it extremely hard to hack into but having the database the website relies on running unprotected on the same or a 2nd box with a standard default login granting administrative privileges.

Whether there is justification for a high-security hosting facility depends on the liability defined in the contracts and possible amounts of money in compensations to be paid in case anything goes wrong. Housing the static company homepage may have different requirements than a credit card processing service. A fake news announcement placed by an attacker on the company website bears different risks as well: For the homepage of the bakery down the street, a faked news announcement may only be a small annoyance. For a stock-based company, a proclamation of bankruptcy could become a self-fulfilling prophecy.

The only way to reliably determine if, when and where to raise the levels of security is by answering some what-if questions: What can happen, what damage will it cause and how much money is needed to protected against it? When creating this list, you will realize some security problems are vital to fix while others, though annoying, don't cause enough harm to be worked on – as scary as that may sound. While making such a list is not hard per se, it requires knowledge about what is possible and how to calculate your risk. Knowledge which could be gained by training and learning, but that would require spending money on security to begin with...

ARNE BLANKERTS is the head of development at NonFood advertising agency in Hamburg, Germany. Arne is the inventor and lead developer of an open source site system "fCMS" (*http://fcms.de*) that is written in object-oriented PHP and makes heavy use of XML. Furthermore he is a trainer, regular author writing articles for the German PHP Magazine, co-authoring books and speaking at (PHP-) conferences. In the unusual case of spare time, he helps maintaining the German translation of the PHP manual.

Thinking in Circles

by Marco Tabini

I have a friend who works with embedded systems. As a software architect, I envy him. Badly. As a business person, I envy him even more.

Embedded systems can be very challenging to work with. For example, one of the systems this friend of mine has worked on was required to collect seismic data for up to four months at a time in an Antarctic station on a single battery pack, and in temperatures that sometimes exceed -50°C (on the wrong side of the Celsius scale). I'll let you imagine the contortions his team has to go through to make that happen.

On the other hand, however, he has the privilege of working with technology that is well understood. The limitations that he has to deal with today—power consumption, computational capabilities, memory and so forth—are nothing compared to the constraints within which applications were required to run as recently as fifteen years ago.

Most of all, however, I envy him because he gets to write his application *once*.

Today's web development environment is a shambles. For all practical purposes, every single bit of code has to be written and tested on a combination of no less than six different platforms (Windows and OSX on hand and Safari, Internet Explorer and Firefox on the other) that are completely at odds with each other. Worse than that, web development itself is performed with a purpose that is at odds with the very basic premise under which the Web was invented.

HTML is *clearly* a data description language, but developers were using it for layout, because that's what users wanted from it, so CSS was invented. Now, I have been in the publishing industry for at least ten years, and I freely admit that CSS is a complete mystery to me—I have learned to use it and understand its quirks, but, frankly, I have always found its design principles mind-boggling.

Then, of course, we have to worry about the fact that, really, HTML and CSS *don't exist*, because each browser vendor has decided, in their infinite wisdom, that a common set of functionality would be inappropriate. Unfortunately, there are compelling arguments for supporting all three browsers, which adds to the overall heartburn to which web designers are subject.

I could go on, of course (don't even get me started on the server side!), but the bottom line is that web development is very much like trying to fit a square peg into a round hole, only to find out that the square peg has sympathetic high explosive charges wired to every side, and the round hole is really just a black circle painted on concrete flooring.

This is worrisome for a number of reasons. First, it stifles creativity—budgets are a finite resource, and the more money spent on making sure that a web application runs on as many platforms as possible is money that is not spent on coming up with new ideas and products. Second, we just keep building hacks on top of other hacks—and that spells trouble for the long-term stability and viability of applications.

What's the solution? It's hard to tell. Vendors seem to be clear on the fact that interoperability beyond the most basic of levels is really of no major concern to them. Ultimately, this could simply give rise to a sort of "super-platform" that makes it possible for developers to completely bypass HTML and company and finally write code once that will run anywhere. Some companies, like Adobe with Flex and Microsoft with SilverLight, are obviously after this very solution, but they have also run into a number of problems like considerable inertia in consumer adoption, limited functionality, paranoia-induced security restrictions, and so forth.

Ultimately, the great risk that we're all running is that the true cost of developing for the web—which is already fairly high today—will reach a point where investment capital will simply shift elsewhere.

ElePHPants!

The elePHPants love to travel the world! Here are some photos that they have sent us from their travels.

Move over Waldo, Where's Elephpant? Photo courtesy of Rein Velt.

Rumble in the jungle. Photo courtesy of Marcia Ringel and Alan Seiden.

Do you have photos of your ElePHPant? We would love to print them. Send us a high-resolution image (or if you have the high-resolution image posted online, just send us the link) of your ElePHPant. We'll choose several photos a month to publish.

NEXCESS.NET
INTERNET SOLUTIONS

NEXCESS.NET Internet Solutions
304 1/2 S. State St.
Ann Arbor, MI 48104-2445
http://nexcess.net

PHP / MySQL SPECIALISTS!

Simple, Affordable, Reliable PHP / MySQL Web Hosting Solutions

POPULAR SHARED HOSTING PACKAGES

MINI-ME $6.95/mo

- 500 MB Storage
- 15 GB Transfer
- 50 E-Mail Accounts
- 25 Subdomains
- 25 MySQL Databases
- PHP5 / MySQL 4.1.X
- SITEWORX control panel

SMALL BIZ $21.95/mo

- 2000 MB Storage
- 50 GB Transfer
- 200 E-Mail Accounts
- 75 Subdomains
- 75 MySQL Databases
- PHP5 / MySQL 4.1.X
- SITEWORX control panel

POPULAR RESELLER HOSTING PACKAGES

NEXRESELL 1 $16.95/mo

- 900 MB Storage
- 30 GB Transfer
- Unlimited MySQL Databases
- Host 30 Domains
- PHP5 / MYSQL 4.1.X
- NODEWORX Reseller Access

NEXRESELL 2 $59.95/mo

- 7500 MB Storage
- 100 GB Transfer
- Unlimited MySQL Databases
- Host Unlimited Domains
- PHP5 / MySQL 4.1.X
- NODEWORX Reseller Access

interworx: CONTROL PANEL

All of our servers run our in-house developed PHP/MySQL server control panel: INTERWORX-CP

INTERWORX-CP features include:

- Rigorous spam / virus filtering
- Detailed website usage stats (including realtime metrics)
- Superb file management; WYSIWYG HTML editor

INTERWORX-CP is also available for your dedicated server. Just visit http://interworx.info for more information and to place your order.

WHY NEXCESS.NET? WE ARE PHP/MYSQL DEVELOPERS LIKE YOU AND UNDERSTAND YOUR SUPPORT NEEDS!

NEW! PHP 5 & MYSQL 4.1.X

php 5 — MySQL 4.1.x

We'll install any PHP extension you need! Just ask :)

PHP4 & MySQL 3.x/4.0.x options also available

php 4 — MySQL 3.x/4.0.x

128 BIT SSL CERTIFICATES

AS LOW AS $39.95 / YEAR

DOMAIN NAME REGISTRATION

FROM $10.00 / YEAR

GENEROUS AFFILIATE PROGRAM
UP TO 100% PAYBACK PER REFERRAL

30 DAY
MONEY BACK GUARANTEE

FREE DOMAIN NAME
WITH ***ANY*** ANNUAL SIGNUP

ORDER TODAY AND GET 10% OFF ANY WEB HOSTING PACKAGE
VISIT HTTP://NEXCESS.NET/PHPARCH FOR DETAILS

Dedicated & Managed Dedicated server solutions also available

Serving the web since Y2K