www.phparch.com

php architect

The Magazine For PHP Professionals

Pluggable Authentication Modules

Use PAM to standardize user authentication

Practical Active Record in PHP

Simplify data access

MySQL Babel

Make SQL simple for non-English speakers

FileMaker for PHP Developers

Part II: Make data editing a snap

LIMITED SEATING AVAILABLE



RESERVE YOUR SPOT & SIGN-UP NOW

Test Pattern

Deploying PHP applications

INSIDE:

SECURITY CORNER: Why you should upgrade to PHP 5.2.1 php|news exit(0);





Chicago, May 16-18, 2007

Innovate With PHP

The theme for this year's edition of php|tek is PHP::innovation()—we'll focus on practical examples of how PHP can help developers and companies everywhere build better products faster, less expensively and more securely.

To sign up and for more information, visit:

http://www.phparch.com/tek

Rasmus Lerdorf – Keynote, Andrei Zmievski –
Unicoding with PHP 6, Aaron Wormus – Upgrading
to PHP 5 with Style, Chris Shiflett – PHP Security by
Example, Christian Wenz – Web Security by the
Book, Davey Shafik – Everything but the Code,
Derick Rethans – Exposing Hidden PHP Secrets, Ilia
Alshanetsky – High Performance PHP, Jason Sweat
– Test Driven Development, Jeff Moore – Writing
Maintainable PHP Code, Joe Stagner – Programming
with Microsoft AJAX, John Coggeshall – Applications
Build in a Snap using Zend Framework, Lukas Smith
– Lock Your Database Application!, Paul Reinheimer
– Zend PHP 5 Certification Crash Course,
Wez Furlong – PDO, and many more...

php | architect™

CONTENTS



FEATURES

8 Practical Active Record in PHP

Simplify data access

18 Pluggable Authentication Modules

Use PAM to standardize user authentication

28 MySQL Babel

Make SQL simple for non-English speakers

38 FileMaker for PHP Developers

Part II: Make data editing a snap

by Dirk Merkel

by Mikael Johansson

by Alessandro Rosa

by Jonathan Stark

COLUMNS

4 EDITORIAL

That's what parents are for...

6 php|news

46 TEST PATTERN

Deploying PHP Applications

51 SECURITY CORNER:

Why you should upgrade to PHP 5.2.1

57 exit(0);

Is Google Really the Best We Can Do?

by Sean Coates

by Eddie Peloke

by Jeff Moore

by Ilia Alshanetshy

by Marco Tabini

Download this month's code at: http://www.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!

That's what parents are for...

y mind continually attempts to play out different scenarios in which we could drastially improve PHP. For example, I've been thinking a lot about the PHP community and how it relates to PHP's future, lately.

If you read the PHP internals list, you've probably noticed that over the past year, especially, the signal-to-noise ratio has tapered from nearly crystal clear, to occasional bits of clarity mixed with more than a healthy dash of static. I think this is directly related to PHP's continued growth and increased accessibility.

More and more smart people are starting to use PHP. This, ultimately, is a good thing for PHP and the PHP community (obviously), but one of the problems that these smart people bring to the table is one of custom and tradition.

These intelligent folks have switched over to PHP from another platform, perhaps, and often hae a respectable amount of experience on their previous platform. Unfortunately, they're not familiar with The PHP Way, which I've discussed at length in the past. They attempt to apply their past experience to PHP, and often fail—not necessarily because their ideas are bad, but for other reasons such as: "that simply won't work in PHP because of dynamic typing", "10+ years of PHP history dictates that if we do this, we'll break every application in existance", and "that's simply too magical."

These suggestions for the improvement of PHP, while well-intentioned, often spiral into a mailing list torrent of posts. Much of the time, PHP's main influencers simply don't have time to read threads that have posts numbering in the hundreds, and what could have blossomed into a good idea is often temporarily (and sometimes permanently) put on hold. That said, many of the proposed ideas are *not* actually good ones.

This is one of the reasons that I've been thinking about how PHP needs some sort of parental guidance. Perhaps we need to collectively form a group of wise PHP contributors that we trust with our language; a group that would form roadmaps (informal, even), veto feature requests, and serve as the ultimate body that controls PHP's future.

I'm sure some of you squirmed when reading that last paragraph. I'm equally uneasy about uprooting the whole system of how PHP has evolved (admittedly very successfully) over the past 10+ years. Is it time for change, though? I think so.

I love the idea of forming a group that will guide PHP, but how can we do that without alienating the hundreds of people who have contributed to PHP in the past, and are either no longer active, or who haven't contributed enough to cross \$magicalThreshold and be considered for membership in this controlling body?

This is definitely a social problem, and we, the PHP community, aren't very good at solving these, it seems. We're great at solving technical issues, but like many excellent developers, our people skills need some polish.

I don't have a solution to the problem of PHP's changing community. There's no simple solution, in my opinion. I would, however, love to hear from readers about their opinions on where PHP's going and how we can optimize its future.

Jean Contra

php | architect™

Volume 6 - Issue 3

Publisher Marco Tabini

Editor-in-Chief
Sean Coates

Editorial Team

Arbi Arzoumani Steph Fox

Graphics & Layout

Arbi Arzoumani

Managing Editor

Emanuela Corso

News Editor

Eddie Peloke news@phparch.com

Authors

Ilia Alshanetshy, Mikael Johansson, Dirk Merkel, Jeff Moore, Alessandro Rosa, Jonathan Stark, Marco Tabini

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

Although all possible care has been placed in assuring the accuracy of the contents of this magazine, including all associated source code, listings and figures, the publisher assumes no responsibilities with regards of use of the information contained herein or in all associated material.

php|architect, php|a, the php|architect logo, Marco Tabini & Associates, Inc. and the Mta Logo are trademarks of Marco Tabini & Associates, Inc.

Contact Information:

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

Printed in Canada

Copyright © 2003-2007 Marco Tabini & Associates, Inc. All Rights Reserved



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 developement

Fax: 604.691.1678

php | news

PHP 5.2.1 Released

PHP.net announces the latest release of PHP, version 5.2.1.



"The PHP development team would like to announce the immediate availability of PHP 5.2.1. This release is a major stability and security enhancement of the 5.X branch, and all users are strongly encouraged to upgrade to it as soon as possible."

What's new? Some security enhancements and fixes include:

- Fixed possible safe_mode & open_basedir bypasses inside the session extension
- Prevent search engines from indexing the phpinfo() page.
- Fixed a possible overflow in the str_replace() function.
- Memory limit is now enabled by default.
- Much more...

Grab the latest version from http://php.net//downloads.php.

Zend Platform 3.0 Released

Zend Developer Zone brings word of the release of Zend Platform 3.0:

"Zend Technologies, the PHP company, today announced the general availability of Zend Platform 3.0. Zend Platform 3.0 is the only PHP application server created specifically to support the management, performance, integration and scalability requirements of organizations that use PHP to run business-critical web applications. Zend Platform improves the end user experience with these web applications application enabling better responsiveness, reduced application downtime, and richer functionality."

"PHP is the most successful language

for creating and operating modern, dynamic web applications. Driven by its ease, flexibility, and power, it is increasingly being used by commercial organizations to create business-critical web applications," said Mark de Visser, Chief Marketing Officer at Zend Technologies. "Zend Platform 3.0 complements PHP with capabilities that these organizations need to guarantee the best end user experience and maximum uptime for their web applications."

Get more information from http://www.zend.com.

Smutty 0.0.8

The Smutty team announces the latest revision of their MVC PHP framework. From the smutty project home page:

"Smutty is an MVC web development framework for PHP. Smutty is based around the Smarty templating engine. It is designed to be as developer friendly as possible, doing as much work as it can so you don't have to."

The latest release includes an improved model manager and stabalizing of the API for version 0.1. Check out all the latest stuff by visiting http://smutty.pu-qh.com/.

GD 2.0.34

Pierre-Alain Joye announces the latest release of the GD library.

"It was a long road to get GD released. I like to consider this release as a second born for the libGD project. See the release announcement at http://www.libgd.org/ReleaseNote020034 for all the details."

"This release also marks an official freeze of the 2.0.x tree. Only security related fixes will make it into future 2.0.x releases. The development tree now targets version 2.1.0. Additional information about the roadmap will

come in the next few weeks."

Get the latest copy from http://www.libgd.
org/.



MSLG Multi Switch Looking Glass for PHP 0.1.0

The MSLG team announces their latest release, 0.1.0. What is it? The MSLG home page describes it as:

"... a script for manage ports (open and close) on IOS and CatOS switches. Initially used for an Lan-Party for open a port when a gamer will pay for his place. It's inspired of the MRLG for GNU Zebra and Cisco IOS routers by 2002-2004 Denis Ovsienko."

Need to watch some ports? Visit http://www.qamesover.ch/~mslq/.

Zend Studio 5.5 for i5/ OS

Zend's Developer Zone brings news of the latest release of Zend Studio for i5/OS.

"The latest version of Zend's flagship IDE for PHP development has been released for i5/OS. Zend Studio is an industry leading PHP Integrated Development Environment designed for professional developers, which includes all the development components necessary for the full PHP application lifecycle. Through a comprehensive set of editing, debugging, analysis, optimization, database tools and testing Zend Studio speeds development cycles and simplifies complex projects. Zend Studio Professional for i5/0S is enhanced to work with the integration toolkit provided with Zend Core on i5/0S".



What's new?

- Instant online debugging and error fixing against Zend Core for i5/0S
- PHP 5.2 support
- MySQL on i5/OS support
- Zend Framework Integration
- much more

Get all the latest details from http://www.zend.com.



Grab the latest extension updates from PECL.

PHP Parser 0.2.0

PHP_ParserGenerator exists can be used as a way to generate specific parsers. This parser is customized for usage in metadata extraction, and requires PHP 5.0.0 or newer.

PHP_Parser_DocblockParser 0.1.0

PHP_Parser_DocblockParser is designed for use with PHP_Parser, but is also a general/** docblock */ parser for phpDocumentor-style docblocks.This package is fully unit-tested, and is based on a PHP_ParserGenerator-generated parser.

Validate LV 1.0.ORC1

Data validation class for Latvia. Provides methods to validate:

- VAT number
- Registration number
- Swift code
- Telephone number
- Person ID
- IBAN Bank account number

for Latvian Banks

- Postal code
- Passport
- Person name



Check out some of the hottest new releases from PEAR.

PHP Beautifier 0.1.13

This program reformats and beautifies PHP 4 and PHP 5 source code files automatically. The program is Open Source and is distributed under the terms of the PHP License. It is written in PHP 5 and has a command line tool.

PEAR PackageUpdate 0.6.1

PEAR_PackageUpdate (PPU) is designed to allow developers to easily include auto updating features for other packages and PEAR installable applications. PPU will check to see if a new version of a package is available and then ask the user if they would like to update the package. PPU uses PEAR to communicate with the channel server and to execute the update.

PPU allows the end user to take some control over when they are notified about new releases. The PPU Preferences allow a user to tell PPU not to ask about certain types of releases (bug fixes, minor releases, etc.), not to ask about certain release states (devel, alpha, etc.), not to ask until the next release or not to ask again.

PPU is just an engine for package updating. It should not be used directly. Instead one of the driver packages such as PEAR_PackageUpdate_Gtk2 should be used, depending on the application or other package.

HTML QuickForm altselect 1.0.RC1

A QuickForm plugin that extends the select element and turns its options into checkboxes or radio buttons depending on whether the multiple html attribute was set or not. For extra options not listed, you can also render an **Other** textfield.

pearweb phars 1.1.0

Separate sub-package just for the .phars, to cut down significantly on the size of the pearweb package.

PDO IBM 1.1.0

The PDO_IBM extension provides an IBM database driver for PDO. This driver supports IBM DB2 Universal Database, IBM Cloudscape, and Apache Derby databases.

timezonedb 2007.2

timezonedb is a drop-in replacement for the builtin timezone database that comes with PHP. You should only install this extension in case you need to get a later version of the timezone database than the one that ships with PHP.

ibm db2 1.6.0

This extension supports IBM DB2 Universal Database, IBM Cloudscape, and Apache Derby databases.

pecl http 1.5.0 RC2

This HTTP extension aims to provide a convenient and powerful set of functionality for one of PHPs major applications. It eases handling of HTTP urls, dates, redirects, headers and messages, provides means for negotiation of clients preferred language and charset, as well as a convenient way to send any arbitrary data with caching and resuming capabilities.

It provides powerful request functionality, if built with CURL support. Parallel requests are available for PHP 5 and greater.

Practical Active Record in PHP

The Active Record design pattern provides an objected oriented wrapper around database access. In this article, we'll cover how I analyzed existing Active Record implementations and found myself creating a powerful tool for use in PHP/MySQL application development.

by Dirk Merkel

PHP: 5.1+ --with-pdo-mysql

0/S: Any supported by PHP

Other Software: MySQL (any)

TO DISCUSS THIS ARTICLE VISIT:

http://forum.phparch.com/362

Imost every project I have worked on uses a relational database for data storage. Whether I start a new project or perform maintenance on an existing installation, I have to give some thought to the mechanics of saving and retrieving information from the database. Wouldn't it be nice if I had a tool that would allow me to focus on my application's business logic without having to worry too much about how to create, read, update, and delete rows from the tables in my database?

The Active Record Design Pattern

Luckily, much smarter people than myself have already considered this need for a tool, and have come up with a blueprint for it: the Active Record design pattern. Design patterns are generalized solutions to often recurring problems in programming. In this case, manipulating rows of a database table is the problem, and the Active Record pattern is the solution.

Astute readers will have noticed the acronym CRUD

(Create, Read, Update, Delete) hiding at the end of the first paragraph. CRUD is often mentioned in connection with the Active Record pattern, because that is precisely the functionality it provides. Active Record typically consists of a class that wraps a row of a database table. By invoking methods on the object, we can get it to issue corresponding SQL statements to the underlying database.

Let's look at an example. Assume that we have a MySQL table, employees, as defined by the SQL statement in Listing 1. Now take a look at the code snippet in Listing 2 to illustrate how an Active Record object might allow you to transparently manipulate the database.

As you can see, individual columns are conveniently accessible as members of the object. The <code>isValid()</code> method checks each of the fields to make sure we will not encounter any surprises when saving the object to the database. The <code>save()</code> method alternatively creates a new row or updates an existing one. The <code>findByPrimaryKey()</code> method allows us to retrieve a row using our underlying table's primary key field, id. Finally, the <code>delete()</code>

method lets us easily delete a row from the table.

By now you are probably getting a pretty good idea of how valuable a tool a well-written and flexible implementation of the Active Record pattern can be. Within a dozen lines of code, we have inserted, selected, updated, and deleted (essentially the SQL equivalent of CRUD) from our database. All the details of writing queries and data validation have been hidden behind a nice, shiny, object-oriented exterior.

At this point, you are probably thinking that this isn't much different from what you have been doing all along. It appears that the Employee class is simply a class that is able to save its state to the database. Actually, you'd be correct in assuming the Employee class in Employee. php is empty:

```
class Employee extends ActiveRecord
{
}
```

In other words, all database interaction is handled simply by extending ActiveRecord and adding your business logic as necessary.

The Active Record pattern isn't anything new. Martin Fowler described it in his book "Patterns of Enterprise Application Architecture", published in 2002. (By the way, I encourage you to read anything Mr. Fowler has published.) I think it is fair to say that Mr. Fowler can be credited as one of the people to recognize and document this design pattern that many programmers were applying over and over again. Moreover, design patterns are not limited to any particular programming language. Since the problem of storing and retrieving data is a general one, it comes up in most programming languages, although the implementation of the pattern differs between programming languages and from developer to developer.

More recently, the Active Record pattern has been implemented in CakePHP, an open source development framework, and even more famously in Ruby on Rails. The creators of the Rails framework have done a fine job at elevating the Active Record pattern to the status of a power tool no developer should be without. In creating my own Active Record implementation in PHP, I took inspiration from some of the design choices of the Rails implementation. The rest of this article will focus on describing my own choices and solutions in creating this implementation.

Adding More Powerful Features

I started with a list of features I wanted to see in my ActiveRecord class:

Individual fields should be available as members of the class

- Basic create, read, update and delete functionality
- Data validation methods should be available for all members and each member individually
- Flexibility: little or no required configuration, but default options can always be overwritten
- Extensibility: using the ActiveRecord class in different projects should be painless and adding business logic to the individual classes should be natural
- Convenience methods for finding a particular record using a variety of parameters
- It should be database agnostic. In other words, we should be able to use the same tool with a different database engine with a minimal amount of code changes

This list is by no means complete, but it should serve as a solid base for further expansion. I wanted to avoid feature creep for the time being.

LISTING 1

```
CREATE TABLE 'employees' (
   id' int(10) unsigned NOT NULL auto_increment,
   'first_name' varchar(100) NOT NULL default '',
   last_name' varchar(100) NOT NULL default '',
   department_id' int(10) unsigned NOT NULL default '0',
   'ss_number' char(10) NOT NULL default '',
   'date_updated' timestamp NOT NULL default '0000-00-00 00:00:00' on update
   CURRENT_TIMESTAMP,
   PRIMARY KEY ('id')
) ENGINE=MyISAM DEFAULT CHARSET=latin1;
```

LISTING 2

```
<?php
   require_once('Employee.php');
 5 $dbh = new PDO('mysql:host=localhost;dbname=mydatabase', 'user',
 'password');
   // create and save an employee object
 8 $employee = new Employee($dbh):
 10 $employee->last_name = 'Doe';
11 $employee->department_id = 1;
12 \text{$employee->ss_number = '123-45-678';}
14 // validate each of the fields
15 if ($employee->isValid()) {
16 $employee->save();
       echo "Please fix the following errors:\n";
       implode("\n", $employee->validationErrors);
22 // now let's retrieve a different row and modify it
23 $employee->getByPrimaryKey(10);
24 $employee->last name =
26 $employee->save();
28 // finally, let's delete a row
29 $employee->getByPrimaryKey(20);
30 $employee->delete();
```

With all this talk of classes and objects, you have probably figured out that I chose to implement my version of Active Record in PHP 5. Actually, there are a couple of features that require PHP 5.1.x. You will also need a database—MySQL, if you want to try the sample code for yourself. Finally, you will need PDO support. PDO is a lightweight access layer for databases in PHP 5.

I decided that the ActiveRecord class would essentially be a container for the database fields. In addition, it would be responsible for providing an interface for access to any other methods in the supporting classes, which would be hidden, proxied, and instantiated automatically. In other words, using the final ActiveRecord class should never require the programmer to instantiate supporting objects, or to descend the object hierarchy to get to the desired method call. For example, although the validate() method belongs to the DataManager object, it is never necessary to call it like this: \$employee->data-Manager->validate(). Instead, \$employee->validate() will be enough because the Employee object knows to call the validate() method on the DataManager object and handle the return values correspondingly.

Something else that was important to me was that I wanted to separate the generic data structure from the nitty-gritty details of supporting a specific database engine. Furthermore, since I wanted to support multiple database engines and drivers, it made sense to create a separate data manager class for each combination of engine and driver. Not surprisingly, the various data managers share a great deal of functionality that I was able to extract into the abstract parent class, DataManager.

Assembling these components gave the outline of a design that can be summarized in the class diagram in Figure 1. The ActiveRecord class works by extending it. As you will see later, you will have the ability to overwrite some of the defaults when extending the ActiveRecord class, but the concept is the same—even for table schemas that are much more complicated.

The ActiveRecord Class

Let's take a look at the ActiveRecord class in more detail. For the following discussion, please refer to the code in Listing 3 (found in this month's code package). Also, for brevity's sake, I will be omitting discussion of some of the less important methods and members.

Here is a summary of the members of the class:

- \$dbh: A reference to the database connection object that was initially supplied to the ActiveRecord constructor
- \$tableName: The name of the table whose rows we are wrapping
- \$primaryKey: An array of the field names that

- comprise the primary key of the table
- \$autoIncrement: The name of the first (if any) auto-increment field in the table
- \$autoValidate: A Boolean flag indicating whether to validate all fields before the save() method is called
- \$dataManager: A reference to the underlying data manager object
- \$newRecord: A Boolean flag indicating whether the instance represents a new row or an existing one
- \$validationErrors: An array of error messages generated during validation
- \$fields: An array of the fields of the database table. The array also tracks Boolean modification flags. Example:

Now let's examine the more interesting methods of the ActiveRecord class in turn. The constructor, __construct(), takes a database connection as a parameter. This connection is then promptly assigned to the \$dbh member. More importantly, the constructor then calls the dataManagerFactory() method. As the name suggests, this method determines the kind of database connection object that was supplied to the constructor, and tries to instantiate the corresponding DataManager object. For example, when we instantiated a PDO database connection to a MySQL database in Listing 2, the factory method recognized that the parent class was of type PDO and that the driver was mysql. As a result, the dataManagerFactory() instantiated and returned an ActiveRecord_PdoMysqlDataManager object. If a suitable data manager cannot be found, an ActiveRecord_Exception is thrown.

The dataManagerFactory() method is a major contributor to the inherent flexibility of the ActiveRecord class. If you find that you need to support a new database, you only have to create the corresponding data manager by extending the abstract DataManager class and modify the switch statement in the dateManagerFactory() method to recognize the new driver. That way, you might end up with classes such as ActiveRecord_PdoFirebird, ActiveRecord_Mysqli, or ActiveRecord_CustomDbDriver.

One of the reasons that I decided to base this example on the relatively new PDO interface is that it already supports a variety of databases: MS SQL Server, Sybase, Firebird, DB2, Informix, MySQL, Oracle, ODBC, PostgreSQL and SQLite. If you want to support any of these databases, you can easily make a copy of the ActiveRecord_PdoMysqlDataManager class and make the necessary changes to support the data types and SQL dialect

LISTING 4

```
2 /* vim: set expandtab tabstop=4 shiftwidth=4 softtabstop=4: */
  4 * Abstract data manager class.
 6 * This abstract class provides a framework and describes
7 * the interface for all actual data manager sub-classes.
8 * Individual data managers need to be created for each
  9 * database connection & driver.
 11 * PHP version 5.1
 13 * @category
 14 * @package
15 * @author
                    Dirk Merkel <dirk@waferthin.com>
 16 * @copyright 2007 Waferthin Web Works LLC
 17 * @see
                  ActiveRecord
 18 * @since
                    Jan 9, 2007
 19 * @version SVN: $Id$
 21 abstract class ActiveRecord_DataManager
 22 {
 24
         * Instance of a ActiveRecord_DataManager object.
         * We only want to instantiate one DataManager per connection 
 * (PDO object). This array contains a reference to each of the
         * DataManager objects that have been created.
 30
         * @access protected
         * @var array - instances of DataManager objects
 33
34
        protected static $instances = array();
 36
         * ActiveRecord object reference.
 38
         * @var object - reference to an ActiveRecord object
 39
 40
        protected $activeRecord:
 42
 43
 44
         * DB connection object.
 45
 46
         * @access protected
         * @var object - reference to DB connection object
 49
        protected $dbh:
 50
         * Array of associative arrays with column data indexed
 53
54
         * by dbName::tableName::primaryKey.
 55
56
         * @access protected
         * @var array - meta-data for the table(s)
 58
        protected $columnData = array();
 60
         * Validation error message.
         * @access protected
         * @var string - error message generated during validation
 65
 66
        public $validationError = null;
 67
 68
         * Array of irregular English plurals.
         * @access protected
         * @var array - singular-plural mapping
 73
74
75
76
77
78
79
         protected $plurals = array(
                                        'person' => 'people',
                                                     => 'men'
                                         'man'
                                         'woman'
                                                     => 'women
                                    );
 80
         * DataManager constructor.
          * @access private
 84
          * @param ActiveRecord activeRecord a reference to ActiveRecord
object
          * @return void
```

LISTING 4: Continued...

```
private function __construct(ActiveRecord $activeRecord)
 88
 89
            $this->activeRecord = $activeRecord;
 90
            $this->dbh
                              = $activeRecord->dbh;
 94
         * DataManager destructor.
 95
 96
 97
         * @return void
        public function __destruct()
104
         * Utility method to turn object names
         \ensuremath{^{*}} with underscores into camel case.
         * @access protected
         * @param string text - text in underscore notation
         * @return string - text in camel case notation
        protected function underToCamel($text)
            return\ preg\_replace('/\_([a-z])/e',\ "strtoupper('\\1')",\ \$text);
118
         * Utility method to turn object names
         * with camel case into underscores.
         * @access protected
         * Oparam string text - text in camel case notation
         * @return text - text in underscore notation
124
        protected function camelToUnder($text)
126
127
            129
            return strtolower($text);
130
         * Pluralize English names.
         * This method tries to construct the grammatically
         * correct plural of an English word using commo
136
         * grammar rules and a list of irregular plurals.
139
         * @param string name - singular word
         * @return string - plural of input word
141
        protected function pluralize($name)
144
             / check whether this is a known irregular plural
146
            if (array_key_exists($name, $this->plurals)) {
           return $this->plurals[$name];
} elseif (preg_match('/y$/', $name)) {
   return preg_replace('/y$/', 'ies', $name);
} elseif (preg_match('/s$/', $name)) {
                return $name . 'es';
            } else {
                return $name . 's';
        }
         * (Re-)initializes ActiveRecord object.
         * and should be run upon first instantiating the object
         * or when the delete function has been called.
164
         * @access protected
         * @return void
165
        protected function resetActiveRecord()
             // initialize the values of the fields
            $newFields = arrav():
            foreach (array_keys($this->columnData[$this->activeRecord-
>tableNamel) as $field) {
```

spoken by the database of your choice.

The constructor also takes an optional second parameter, \$primaryKey. This is essentially a convenience function, allowing the programmer to instantiate the object and load a row from the database at the same time. If the primary key is supplied, the constructor calls the findByPrimaryKey() method of the ActiveRecord object. Don't worry if you cannot find the findByPrimarykey() method in the ActiveRecord class. As we will see later, this is just one of many functions added to the class via method overloading.

Next we have the init() method, which is another convenience function. This allows the programmer to assign to the various fields of the underlying row by supplying an associative array, instead of having to assign each field individually. More specifically, instead of:

There is Magic in PHP

The remaining three methods that complete the ActiveRecord object are the magic methods __get(), __ set() and __call(). Despite the name, there really isn't anything magical about these methods. __get() will be called whenever an attempt is made to access a member (in the case of __get() and __set()) or method (in the case of __call()) that does not exist.

I use the __get() method to provide access to the protected members of the class. More interestingly, I also use __get() to provide access to the fields of the row we are manipulating. Not only does this allow for shorter notation when accessing field values, it also reads much more naturally, and satisfies our requirement that database fields be available as members of the class. For example, let's say we are trying to retrieve the first_name field from our Employee object. The call \$employee->first_name would result in the following sequence of events:

PHP sees that the Employee object does not have a member called first_name; therefore, it invokes the __get() method with first_name as an argument. __get() finds that first_name is a key in the \$fields array, and returns \$this->fields['first_name']['value'].

This mechanism allows us to modify the \$fields array at will, and have all fields directly accessible as members of the object.

As you may have guessed by now, the __set() method

does exactly the opposite of __get(); it lets the programmer assign values to class members and the fields comprising the database row. There are two things worth noting here. First, if a field is being assigned a value, a corresponding Boolean flag is set. This way, we can always check whether any changes have occurred that would require us to call the save() method to persist these changes to the database. Second, we don't allow access to private members. Not all members should be accessible to the programmer, and this is a way of simplifying the interface of our ActiveRecord class.

For an example of the magic <u>__get()</u> and <u>__set()</u> methods, you only have to look at this line in Listing 2:

```
$employee->first_name = 'John';
```

This call was routed through __set() to store the literal John in \$employee::fields['first_name']. Conversely, the line:

```
echo $employee->first_name;
```

would cause the __get() method to return the value of \$employee::fields['first_name'].

Virtual Methods

Lastly, the __call() method handles all function calls that are not explicitly defined for the ActiveRecord class. There are two basic types of function calls supported by this method.

First, there are method calls that are meant for the underlying data manager. Using reflection, we check to see whether the method name exists in the data manager and whether the method has public visibility. If both requirements are satisfied, we call the data manager's method. Note how we force validation of all fields before saving them to the database. This is another option that can be disabled, as you will see later.

The second kind are those that are being added dynamically to the ActiveRecord object. At this point there are three groups of methods. All of these are available for all the fields of the underlying database row:

- isValid<Field> returns a Boolean value to indicate whether a specific field passes the data type check of the underlying data manager. For example, \$employee->first_name should not be longer than 100 characters because it was defined as varchar(100)
- isModified<Field> returns a Boolean value to indicate whether the field has been changed since it was last retrieved from the database
- findBy<Field>[_And_][_Or_]<Field>] allows you to search the database on any of

the fields. Logical constructs, such as AND and OR, can be used. For example, you can search for employees by matching either first or last name: \$employee->findByfirst_name_Or_last_name('John', 'Doe'). Note that only the fields from the first row of the result set will be loaded into the object!

Due to member and method overloading, the actual interface that ActiveRecord provides is quite different from that you would expect by looking at the method names directly. Refer to Figure 2 to see a class diagram of the public interface exposed by the ActiveRecord class.

Finally, I felt that the ActiveRecord class should have its own exception, which is easily achieved in *Exception*. *php* with another empty class:

```
class ActiveRecord_Exception extends Exception \{ \footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\footnote{\fo
```

The Abstract DataManager Class

Now that we have a good grasp of the ActiveRecord class, let's take a look at the way the actual database interaction is handled.

LISTING 4: Continued...

```
$newFields[$field] = array('value' => null, 'modified' =>
false);
            $this->activeRecord->fields = $newFields:
176
177
            $this->activeRecord->newRecord = true:
178
       }
180
181
        * Reset the 'modified' flag for each field.
        * @access protected
         * @return void
186
        protected function resetModified()
187
            foreach ($this->columnData[$this->activeRecord->tableName] as
      => $data) {
$name
                $this->activeRecord->fields[$name]['modified'] = false;
        abstract protected function defineSchema();
195
        abstract public function save(ActiveRecord $activeRecord);
        abstract protected function insert();
        abstract protected function update();
        abstract public function find(ActiveRecord $activeRecord,
$searchString, $args);
        abstract public function delete(ActiveRecord $activeRecord):
        abstract protected function refresh();
        abstract public function isValidField(ActiveRecord $activeRecord,
$field):
208 }
210 ?>
```

As mentioned earlier, the basic interface of the various data manager classes is defined by the abstract DataManager class. This class contains all the logic needed to map the database row to the ActiveRecord, and vice versa. Moreover, only a single instance of a DataManager object per connection is needed to handle any number of ActiveRecord objects. Of course, this fact is hidden because you never explicitly instantiate a DataManager object. All that is done for you automatically by the ActiveRecord class.

You will also note that all the public functions take an ActiveRecord object as the first argument. That is because access to all tables and rows using the same connection is handled by the same instance of DataManager. Each time, we need to supply a reference to the current ActiveRecord object, because no row-specific data is stored within the DataManager instance.

Here is a summary table of the members of the Data-Manager class:

- \$instances: An array of instances of DataManager sub-classes, each of which is using a different database connection
- \$activeRecord: A reference to the ActiveRecord object on which the DataManager is operating
- \$dbh: A reference to the database connection object initially supplied to the ActiveRecord constructor
- \$columnData: Column metadata that each DataManager sub-class has to extract from the corresponding database. The actual data structure, and how to use it, is up to the author of the DataManager sub-class

LISTING 6

```
<?php
 3 class EmployeeProject extends ActiveRecord
        // how to overwrite the default table name
       protected $tableName = 'employees_projects';
       // how to overwrite the default primary key ..
       protected $primaryKey = array('employee_id', 'project_id');
       // how to turn off data validation before DB operations
       protected $autoValidate = false;
       // perform advanced validation beyond
16
17
       public function isValidemployee_id()
18
           if (!in_array($this->employee_id, array(1, 2, 3))) {
19
               $this->validationErrors[] = 'Only employee IDs 7, 8, & 9 are
acceptable';
20
           } else {
               return $this->dataManager->isValidField($this, 'employee_
id');
25 }
```

- \$validationError: An error message string generated by the most recent call to the data validation method(s)
- \$plurals: Associative array of irregular plurals
 of the English language. This is used in an
 attempt to properly pluralize the name of the
 ActiveRecord sub-class to derive the corresponding table name

Let's look at some of the DataManager methods in more detail—see Listing 4. The setup() method tries to derive the table name. The first option is to take the name of the sub-class (Employee in our example) and change it, using the pluralize(), camelToUnder() and underToCamel() utility methods. For example, a class called MyPiece would point to a table called my_pieces. This convention saves a lot of time and effort when used correctly, but—as stated in our functional requirements—conventions can be overwritten. In this case, the programmer can define a \$tableName member in the ActiveRecord sub-class, and it will be used verbatim.

Note that we are using these functions only to convert the name of the class that extends ActiveRecord to derive the underlying table name. We are not using it to convert the individual field names to camel case members of the ActiveRecord object. This is a deliberate decision derived from the fact that we want to wrap the row, not map it. Having said that, there is nothing to prevent you from implementing a mapping in your own DataManager sub-class. For example, the field name first_name could be converted into the member name firstName. I am actually playing with the idea of adding this as an option that can be overwritten in the subclass, similar to the table name. Remember—convention over configuration.

The only two other methods that are implemented in the DataManager class are resetActiveRecord() and resetModified(). The former would be called to initialize an ActiveRecord object or to return it to its original state before a row was loaded. We need to be able to do this because the ActiveRecord object might change identity during its lifetime. Potentially, each time we call any of the find...() functions, a different row can be loaded and most of the members can change. The same holds when we delete a row.

The resetModified() method simply sets the modified flag corresponding to each field to FALSE. As you will see in the implementation of an actual DataManager class, we need to do this whenever we save the object or load a new row.

The remaining methods are abstract in nature. That means that we only define the signature of the function, consisting of the name and input parameters; the actual implemenation is left to the individual DataManager

sub-class. This makes sense, if you think about it, since all these methods are database and driver dependent. Take the save() method, for example. You will have to execute an SQL statement to write the data back to the database. Any SQL can be database dependent. In the same way, the way you connect to the database and ask it to execute your SQL statement will depend on the driver supplied by the user. Both these variables are unknowns, which is why the implementation of these functions must be left to database and driver specific sub-classes of DataManager.

Following is a list of the abstract methods and the functionality each provides:

- defineSchema(): queries the database for metadata of the table fields. It is basically a way to capture information about the table's schema. The information is used to initialize the ActiveRecord object and to validate the data
- save(): writes the data to the database. This
 method is essentially a switch. If the ActiveRecord object represents a new row, the
 insert() method is called. Conversely, if the
 object represents an existing row, update()
 is called
- insert(): writes the ActiveRecord object's data to the database using an INSERT statement
- update(): writes the ActiveRecord object's data to the database using an UPDATE statement
- find(): constructs SELECT statements and initializes the ActiveRecord object with the first of the rows returned. This method does the actual work whenever one of the findBy<Field>[_And_] [_Or_]<Field>] methods is invoked on the ActiveRecord object
- delete(): issues a DELETE query to delete the row represented by the ActiveRecord object
- refresh(): after saving an object, we need to re-read the row from the database because some of the fields, such as auto-increment and timestamp fields, are updated when the query runs
- isValidField(): using the metadata extracted by the defineSchema() method, this method is able to validate each of the fields against the type restraints of the underlying datahase

A Data Manager For MySQL And PDO

Given the abstract data manager, let's look at how we

would go about implementing an actual data manager for a MySQL database using a PDO connection (see Listing 5 in this month's code package). Since much of the functionality has already been covered in the discussion of the parent class, I want to only focus on the most interesting features of this particular sub-class.

Looking at the members of the class, you will notice \$preparedInsert, \$preparedSelectByPrimaryKey, and \$preparedDelete. These members contain references to prepared statements, indexed by table name. I use prepared statements wherever possible because they allow us to take advantage of PDO's automatic escaping when binding variables. Also, we can expect better performance when making repeated use of the same prepared SQL statement. Note that it would not make sense to save references to prepared update or find statements. Update queries can vary greatly, because we are only updating columns that have been modified. Similarly, due to the great flexibility we are giving the programmer in constructing various SELECT queries using logical AND and OR, it does not make sense to store references to these statements. However, since the most common SELECT statement uses the primary key to retrieve a row, we will store a reference to that statement.

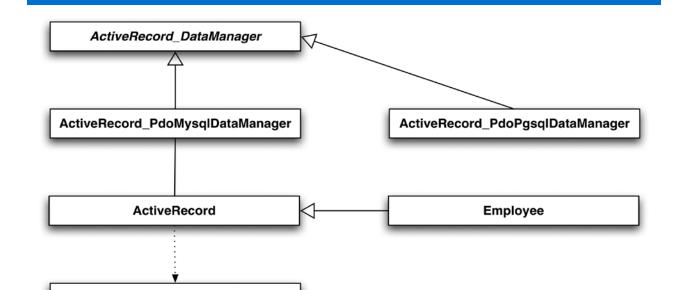
Remember I said that data managers are almost like singletons? You only need to take a look at the getInstance() method to understand what I was referring to. This method gets called from the ActiveRecord object and returns a reference to a DataManager object. What the getInstance() method actually does is to check

FIGURE 1

whether a DataManager already exists for the given database connection. If so, it returns a reference to that object. Otherwise, it creates a new instance using the new database connection, and adds a reference to it to the \$instances member array. Finally, the getInstance() method also handles some setup, which includes calling the methods to retrieve the metadata and to initialize the ActiveRecord object.

There are eleven methods following the naming convention isValid<MySQLDataType>(). Each of these is used to validate a given field's value against the database's type constraints. For example, calling isValidInt() on a field of type mediumint will check whether the value of the field falls outside the minimum or maximum value for that MySQL data type. These validation functions are quite basic, and are primarily designed to preempt any nasty surprises when writing the values to the database. At the moment, some of them don't even support the various formats that are considered acceptable in MySQL. Fortunately, you can overwrite the isValid...() function for any given field (see Listing 6) and make the validation logic as comprehensive as your application dictates.

The last method worth examining is find(). The \$searchString argument passed to the find() method is essentially the name of the find function invoked on the ActiveRecord object. For example, a call such as \$activeRecord->findByfirst_name_And_ss_number will result in the data manager's find() method being called with the \$searchString parameter set to first_name_And_ss_



ActiveRecord_Exception

number. The find() method will then explode the string by _And_ and _Or_, and construct the corresponding SE-LECT query.

Active Record typically consists of a class that wraps a row of a database table.

Revisiting Our Requirements

Now that we have a functioning implementation of the Active Record design pattern, let's revisit our requirement of "convention over configuration" to see how well we have adhered to it so far. If you simply extend ActiveRecord and don't overwrite any of the members, the object will make the following assumptions: First, the name of the table can be derived from the name of the class (i.e.: class Employee wraps table employees). Second, the primary ID of the table is an integer field called id. Third, all fields will automatically be validated before saving the row to the database; this applies to both INSERT and UPDATE statements.

However, any of the above conventions can be easily overwritten. Take a look at Listing 6, where we extend ActiveRecord in a class named EmployeeProject. By overwriting the \$tableName, \$primaryKey, and \$autoValidate members, we are able to let ActiveRecord know that we want to wrap rows from the table employees_projects where the primary key is (employee_id, project_id), and that we do not want automatic validation before saving any record to the database. Furthermore, we overwrote the method isValidemployee_id() to add additional validation logic. Note the way we call the parent class's validation method as well.

If you follow some reasonable conventions as you design the tables in your database, you will be able to simply extend the ActiveRecord class without having to overwrite any of the defaults. At that point, you are free to focus on adding the business logic required by your application without having to worry about database interactions on a micro level.

Looking Ahead

As powerful as this implementation appears, there is plenty of room for improvement. Let's see if we can come up with a list of additional features that would truly elevate the ActiveRecord class to a professional development tool.

- Support for more databases and drivers
- Transaction support—any professional database abstraction layer should support transactions if the underlying database does
- Support for free-form SQL when searching for rows. Although the syntax findBy<member1>_ And_<member2> can be a great timesaver, it lacks power and flexibility. True to our decision to support overwriting of conventions, we need to let the programmer write his own SQL—if s/he wants
- Handling of complete result sets. It doesn't always happen that we know the primary key of the row we need to retrieve. It would be nice if you could manage result sets by sorting, filtering, and paginating ActiveRecord collections

ActiveRecord

FIGURE 2

+dbh: DbConnection

+tableName: string

```
+primaryKey: array
+autoIncrement: string
+autoValidate: bool
+dataManager: DataManager
+newRecord: bool
+validationErrors: array
+field1
+field2
+fieldN
+__construct(DbConnection, string)
+__destruct()
+__call(string, array): mixed
+save()
+delete()
+isValid(): bool
+isValidField1(): bool
+isValidField2(): bool
+isValidFieldN(): bool
+isModifiedField1(): bool
+isModifiedField2(): bool
+isModifiedFieldN(): bool
+findByField1_And_Field2(string, string): bool
+findByField2_Or_FieldN(stirng, string): bool
+getFieldName(): string[]
```

Although we are still within the boundaries of the Active Record design pattern, you might get the feeling that we are straying a bit from the basic implementation. Actually, I would like to take this even further, and add support for composition and relationships.

In real world applications, it rarely happens that your model consists solely of objects that map neatly to individual table rows. More likely, the data needed to instantiate your objects is scattered throughout a handful of tables. Actually, if you go through the process of normalizing your database design (as you should), you will probably make it harder to map data to objects. In a highly normalized database design, you will need multiple joins or queries to retrieve all the required data.

Also, the chances are that various objects contain references to other objects. In our example, an employee should have an address, stored in the addresses table, and the Employee object should contain a reference to an Address object. Our ActiveRecord class currently has no way to reflect that kind of relationship.

The problem of mapping data from objects to a relational database is usually referred to as object-relational mapping, or ORM. On one hand, objects are very good at knowing about relationships between different pieces of data. On the other hand, relational databases don't really have the ability to reflect complex relationships. Granted, we can define triggers and constraints to impose some basic relationships, but object-oriented programming leaves relational databases in the dust when it comes to manipulating related data and complex relationships.

Another thing to consider is that you are not commit-

ted to a relational database as a persistence layer for your objects. You can just as easily write a DataManager sub-class that stores data in XML files, text files or data structures in memory. However, the reason I chose a relational database for storage is that it is by for the most common storage layer I encounter in my daily work.

Summary

By now, we have definitely stepped outside of the boundaries of the basic definition of the Active Record pattern. However, that should not stop our pursuit of the most powerful tool for the job. I plan to make our ActiveRecord implementation smarter by letting it detect and manage relationships between multiple tables and objects. This is something that the Ruby on Rails implementation of Active Record achieves with great elegance.

Isn't that at the heart of being an open source developer? You learn from other solutions, and you keep on adding to your development toolbox.

DIRK MERKEL is a consultant and software architect with a focus on open source technologies. He is Zend certified, and has been working with PHP for 8 years now. He can be reached at <u>dirk@waferthin.com</u>.



Pluggable Authentication Modules

Ever had to rewrite an application to take advantage of your client's authentication scheme. new wished dearly that the service you're installing had Kerberos support? PAM (Pluggable Authentication Modules) is a collection of libraries that handle the authentication tasks of applications, allowing them to stay completely independent underlying the authentication scheme. This article introduces PAM, touches briefly on how to configure it, and explains how to interact with it from within PHP.

by Mikael Johansson

PHP: 4.3. * or better

0/S: Any UNIX-like system

Other Software: PAM, pecl/pam

Useful/Related Links:

http://pecl.php.net/package/pam http://en.wikipedia.org/wiki/Pluggable Authentication Modules

TO DISCUSS THIS ARTICLE VISIT:

http://forum.phparch.com/363

Some time ago, I was asked to write a Web application where users would need to authenticate using their company username and password.

My first problem was that, as with most sizable corporations, the customer had several different user directories, all of which the application needed to support. To make the task more complex, those directories were spread over a large Intranet and used various authentication schemes such as Kerberos, NIS and Active Directory. Since the application was expected to have a lifespan of several years, I also had to deal with the likelihood of its eventually having to support other schemes—some of which might not even be invented yet. The customer would rather avoid the time-consuming and error-prone task of having to modify his application every time an authentication scheme changed, so some kind of ab-

straction was in order.

Fortunately for both of us, this is a problem which has long been solved by separating the application from the actual authentication process. The most common approach involves pushing the details of any specific authentication scheme down into modules that can be loaded and configured at runtime. Using this method I could avoid the need to build support for any specific schemes into the application, while at the same time allowing the customer to swap in future schemes as needed. Looking around for a solution with native support for the required authentication schemes, I found a variety of potential options—just try googling "authentication library" and you'll see what I mean. However, my customer wanted to keep things simple, so the choice fell to PAM.

PAM not only solves the problem beautifully; it also has widespread community support, and is practically a UNIX standard. By choosing a standard solution to handle the authentication process, changes in configuration could easily be undertaken by the customer's own system administrators, who had prior experience of working with PAM.

Come release date, we launched the application configured for NIS and one Kerberos realm running in parallel. Within a year, my client had layered an additional Kerberos realm on top of the previous setup—making for a total of three authentication schemes—without having to change my application one bit.

Pluggable Authentication Modules

First developed in 1996 by Sun Microsystems, PAM is currently supported on several flavours of UNIX including Linux, Solaris, FreeBSD and Mac OS X. Most popular Linux distributions include PAM as part of the default installation, so the chances are high that you have already used it. In the most common configuration, PAM is distributed as a shared library that offers a unified API for accessing multiple lower level authentication schemes, referred to as *PAM modules*. These modules are themselves implemented as shared libraries that can be dynamically loaded into a running program. To use PAM, all an application needs to know is how to invoke the PAM API. Once invoked, PAM will consult a configuration file to transparently load and defer authentication to the appropriate modules.

If you've ever logged onto a Linux system, most likely the login program deferred the actual authentication to PAM, which in turn applied the rules from the configuration file /etc/pam.d/login. If you look at this file, you will almost certainly see that it includes another file, /etc/pam.d/system-auth, which contains the default rules for system authentication. Check out the configuration files for a taste of how PAM works under the hood, but don't worry if it seems a bit complex at first. Later in this article I'll explain something of the configuration format, and teach you how to create your own configuration files.

When writing PAM-enabled applications, in order to maintain flexibility it's important to keep in mind that no assumption can be made regarding how the client will actually be authenticated. That decision must be left entirely to the local administrator. From the perspective of the application developer, PAM should be treated as a black box that will deal with all aspects of user authentication.

Why PAM?

With PAM, the responsibility for deciding the way in

which an application should authenticate is delegated to the local administrator. This frees the developer to tackle issues other than authentication, and gives the systems administrators a higher degree of freedom as to how user validation should be performed. PAM's modular nature allows the administrator to tune the configuration process for all PAM-enabled applications—services in PAM terminology—simply by adding or removing modules. Modules can also be combined or stacked in order to accomplish more complex tasks, such as running multiple authentication schemes in parallel or keeping an audit trail of user logins; but more on this later. Launching a PAM-enabled application is made much easier by the fact that the PAM library is so prevalent in the UNIX world many systems administrators already know how to use and configure it. If there's one thing I've learned from working with sysadmins, it's that they absolutely despise having to compile—and learn all the quirks and gotchas of—a host of custom libraries for every new application they need to install.

Every once in a while, I hear someone mention code reuse and say that we ought to have more of it. Well, PAM is a good example of code reuse through abstraction. Since every authentication module has the same interface, modules can be interchanged transparently. This allows PAM-enabled applications to seamlessly take advantage of the large collection of modules that have accumulated over years of community-driven development. There's support for virtually every authentication scheme in existence; and, since the PAM API is completely open, it never takes long before someone writes a module for any new scheme that pops up. Why roll your own system, when you can reuse a pre-built component that is well encapsulated and has been fully tried and tested? Not to mention the fact that having integrated support for a potential customer's existing authentication system will make your application more attractive to them.

In my experience, all this translates directly to greater marketability, as it allows your application to target a variety of environments without the need for expensive customizations to retro-fit some obscure authentication scheme.

Possible Drawbacks

A possible shortcoming is that PAM deals only in authentication—verifying the identity of a user—and leaves other related tasks, such as authorization or the handling of associated data (e.g. full name, email address, phone number) to the application. For a simple application that only needs to authenticate a username and password this doesn't pose a problem, but once you start building more advanced applications you will want to know more

about the user. My most frequent solution is to wrap PAM in a custom driver that fetches additional user information directly from the user directory, but delegates the authentication process—with all the complexity that entails—to PAM.Listing 1 shows a small example of solving the problem of associated data by coupling PAM to an LDAP directory service using the chain-of-command

There have been a handful of projects that attempt to port PAM to the Windows platform, but the last working release (at http://www.citi.umich.edu/u/itoi/) was for Windows NT only and dated 1998. Most PAM modules would also need to be ported before a Windows version could be truly useful.

PAM is currently supported on most flavours of Unix.

design pattern. Please note that PHP 5 with LDAP support is needed to run this example. The script begins by defining the interface DirectoryService, which all user catalogues or directory services should implement. The authenticate() method validates a username and password and returns TRUE or FALSE depending on the result. The other method, getUserinfo(), is passed a username and returns an associative array containing information about that user. As with PAM, the DirectoryService interface makes no rules about how these methods should actually be implemented.

At the very bottom of Listing 1 is a demonstration of how to create and use the objects. Did you notice how the LdapDirectory instance is wrapped inside an Pam-Auth object? Looking at the PamAuth class, you can see that it indeed tries to authenticate users, but delegates failed authentications—and all getUserinfo() calls—to the next DirectoryService implementation. LdapDirectory, on the other hand, has no idea about how to authenticate users, so its authenticate() method will always return FALSE. It does, however, implement the getUserinfo() method to fetch a given record from an LDAP server.

Don't spend too much time trying to make sense of all the <code>ldap_*</code> functions and how they're used. For all intents and purposes, the whole thing is equivalent to calling <code>mysql_query()</code> and <code>mysql_fetch_assoc()</code> to return a single result row.

Because both classes implement the DirectoryService interface, the resulting chain-of-command is very similar to PAM's principle of *stacking*. This works by building a chain of modules, which is traversed until a module that satisfies the request is found.

The only other shortcoming is that PAM is UNIX only.

Configuring PAM

The PAM configuration format allows for much flexibility as well as complexity, and is largely outside the scope of this article, but I'll explain some of the basics. More detailed information is readily available in the pam.d man page or from a multitude of online guides and howtos. In most setups, the configuration files are stored in the directory /etc/pam.d/ with one file per service; alternatively they may be stored in /etc/pam.conf, in which case each configuration directive is prefixed by the corresponding service name. The examples I will present assume that the configuration is stored in /etc/pam.d/, but converting them for use with /etc/pam.conf should be trivial.

For the sake of module writers—and to ease configuration—PAM separates authentication tasks into four independent groups:

- Account management is responsible for tasks such as verifying that the user is allowed to access the requested service and that the password hasn't expired
- Authentication management takes care of authenticating the user, for example by validating a username and password
- Password management updates the user's authentication tokens. Typically, this group is called upon when changing passwords
- Session management takes care of things that should be done before and after access is granted, for example mounting the user's home directory or keeping an audit trail of logins and logouts

Each configuration file contains a list of PAM modules and the rules by which they should perform the authentication tasks for an associated service. The format of each rule is:

type control module-path module-arguments

The type parameter specifies which management group the rule corresponds to. Valid entries are account, auth, password and session, where each entry refers to one of the previously described groups. The second field, control, determines how PAM will react to the success or failure of the associated module. The most commonly used values are:

- required: an authentication failure will ultimately cause the entire request to fail, but only after any remaining modules have been invoked
- requisite: returns immediately following an authentication failure, without invoking any further modules
- sufficient: the success of the module is enough to satisfy the authentication requirements and return immediately. A failure is not

- considered fatal, and the request is passed to the next module
- include: includes and processes all the rules of the specified type from the configuration file given as an argument. The most common use for this construct is to include system-auth which contains the default system authentication rules

module-path can be either the absolute path to a PAM module or a relative pathname from the default location, which will be either /lib/security/ or /lib64/security/ depending on the system architecture. For example, pam_unix.so would load the shared library that handles standard UNIX authentication.

module-arguments is a space-separated list of module parameters. These parameters are specific to each individual module, although—as you'll see later—most authentication schemes have a few parameters in common.

LISTING 1

```
3 interface DirectoryService {
       function authenticate($username, $password);
       function getUserinfo($username);
 8 class PamAuth implements DirectoryService {
      protected $_next;
       function __construct(DirectoryService $next) {
           $this->_next = $next;
       function authenticate($username, $password) {
           if (pam_auth($username, $password))
               return true;
18
           return $this->_next->authenticate($username, $password);
19
20
      }
       function getUserinfo($username) {
           return $this->_next->getUserinfo($username);
24 }
26 class LdapDirectory implements DirectoryService {
27    protected $_conn;
28
       protected $ basedn:
29
       function __construct($url) {
           $info = parse_url($url)
           $this-> basedn = trim($info['path'], '/'):
33
34
           $this->_conn = ldap_connect($info['host']);
35
           if ('ldap3' == $info['scheme'])
36
               ldap_set_option($this->_conn, LDAP_OPT_PROTOCOL_VERSION, 3);
38
           ldap_bind($this->_conn,
                isset($info['user']) ? $info['user'] : null
39
                isset($info['pass']) ? $info['pass'] : null);
40
41
42
43
       function authenticate($username, $password) {
           return false;
```

LISTING 1: Continued

```
function getUserinfo($username) {
           $query = Idap_search($this->_conn, $this->_basedn,
"(uid=$username)");
           $ptr = null:
           $row = array();
           if (false == ($entry = ldap_first_entry($this->_conn, $query)))
               return null;
           if (false == ($attribute = ldap_first_attribute($this->_conn,
$entry, $ptr)))
               return null:
          do {
               if (false == ($values = ldap_get_values($this->_conn, $entry,
$attribute)))
                   return null;
               if (count($values) <= 2)</pre>
                   $row[$attribute] = reset($values);
               else {
                   unset($values['count']);
                   $row[$attribute] = $values;
66
          } while (false != ($attribute = ldap_next_attribute($this->_conn,
$entry, $ptr)));
           if (!isset($row['dn']))
70
               $row['dn'] = 1dap_get_dn($this->_conn, $entry);
           return $row;
       }
74 }
76 // These classes could be used as:
77 $url = "ldap3://ldap.example.com/ou=people,dc=example,dc=com";
79 $password = "secret";
81 $directory = new PamAuth(new LdapDirectory($url));
82 if ($directory->authenticate($username, $password))
83
       $_SESSION['user'] = $directory->getUserinfo($username);
85 ?>
```

Stacking PAM Modules

Part of the power of PAM lies in its ability to stack multiple modules to accomplish a given task. Some possibilities include using multiple authentication schemes in parallel, or enforcing password sanity checks by stacking cracklib on top of a password-based authentication module such as the standard UNIX authentication mechanism. This fragment from a PAM configuration file illustrates how to stack multiple authentication schemes:

```
auth sufficient pam_unix.so likeauth
auth sufficient pam_krb5.so use_first_pass
auth required pam_warn.so
auth required pam_deny.so
```

Starting from the top, these rules indicate that standard UNIX authentication should be tried first. In most cases,

LISTING 2

```
#%PAM-1.0
# This stack is called when authenticating a user
auth
         sufficient pam_unix.so likeauth nodelay
auth
         required
                     pam warn.so
                     pam_deny.so
auth
         required
# This stack would be called when changing a password
password sufficient pam_unix.so md5 shadow nodelay
                     pam_warn.so
password required
password required
                     pam_deny.so
# This stack is called to ensure that the account is valid
account sufficient pam_unix.so
account required
                    pam_deny.so
# This stack is called before and after granting access
session required pam_unix.so
```

LISTING 4

```
define(PAM_PROXY_PATH, "sudo /usr/bin/php /path/to/pam_proxy.php");
 4 function pam_proxy(\$input, &\$error) {
         $result = false;
        $spec = arrav(
            0 => array("pipe", "r"),

1 => array("pipe", "w"),

2 => array("pipe", "w"));
        if (false !== ($pd = proc_open(PAM_PROXY_PATH, $spec, $pipes)) &&
12
13
14
15
             fwrite($pipes[0], $input)) {
             fclose($pipes[0]);
             if (false === ($status = fgets($pipes[1])))
             $error = "Failed to read status";
else if ("2" != substr($status, 0, 1))
18
19
20
21
                  $error = $status;
             else
                  $result = true:
             fclose($pipes[1]):
24
             fclose($pipes[2]);
25
26
27
28
         else {
             $error = "Failed to execute command '".PAM_PROXY_PATH."'";
29
30
        proc_close($pd);
         return $result;
32 }
34 ?>
```

this involves comparing a hash of the password to one stored in the local /etc/shadow file. The purpose of the likeauth parameter is somewhat obscure and requires an understanding of PAM's internal workings which I won't go into here, but it's a safe bet to always include it when using pam_unix.so.

If the first authentication attempt succeeds, the sufficient keyword causes PAM to immediately return SUCCESS. Should authentication fail the request is passed to the second module, which will try to authenticate against a Kerberos server. In this case the use_first_pass keyword instructs the Kerberos module to use the password stored from the first attempt, rather than prompt the user to supply it again.

In the case where all authentication modules fail

LISTING 3

```
#!/usr/bin/php
  4 if (!extension_loaded("pam"))
 die("500 PAM extension not loaded\n");
fif (false === ($fd = fopen("php://stdin",
        die("500 Internal Server Error\n");
 9 while (!feof($fd)) {
       if (false === ($line = fgets($fd)))
            die("400 Bad Request\n");
       switch (trim($line)) {
            case "auth":
                 if (false === ($username = fgets($fd)) ||
16
                      false === ($password = fgets($fd)))
                     die("400 Bad Request\n");
19
                 $username = trim($username, "\n");
                 $password = trim($password, "\n");
                 $error = null;
                 if (pam_auth($username, $password, $error))
                     print "200 OK\n";
                 else
                     print "401 ".strtr(substr($error, 0, 1024), "\r\n",
26
   ")."\n";
28
29
            case "chpass":
                 if (false === ($username = fgets($fd)) ||
                     false === ($oldpass = fgets($fd)) ||
false === ($newpass = fgets($fd)))
                     die("400 Bad Request\n");
34
                 $username = trim($username, "\n");
$oldpass = trim($oldpass, "\n");
$newpass = trim($newpass, "\n");
36
37
                 $error = null:
40
                 if (pam_chpass($username, $oldpass, $newpass, $error))
41
                     print "200 OK\n";
42
                     print "401 ".strtr(substr($error, 0, 1024), "\r\n",
   ")."\n";
44
                 hreak:
45
46
            case "quit":
47
            case "exit":
                 break 2;
            default:
                 die("400 Bad Request\n");
53 }
55 fclose($fd);
```

the request is passed to pam_warn.so, which writes the failed attempt to the system log. The final rule invokes pam_deny.so, a special module that will always fail authentication. When coupled with the required keyword, this will cause the entire request to fail.

PHP Integration

So how does PHP fit into all this? Well, PAM is implemented as a C API with a shared library. Authentication schemes are also made available as shared libraries that can be dynamically loaded on demand. To be able to take advantage of PAM from within PHP a wrapper extension is needed, and this is where the PECL module pecl/pam comes into the picture. It encapsulates some rather complex PAM interactions in order to provide two simple functions:

```
bool pam_auth(username, password '',&error'])
bool pam_chpass(username, oldpass, newpass [, &er-
ror])
```

The first function, pam_auth, attempts to authenticate a user by passing the supplied username and password to PAM, which in turn loads and invokes the appropriate modules. If the request succeeds it returns TRUE, otherwise the function populates the optional error parameter with a user-readable error message and returns FALSE. The error message provides information about what went wrong down in the authentication scheme, so you might want to log it using error_log() or even echo it out to the user. This PAM interaction uses the modules from the auth stack.

Similarly, pam_chpass uses PAM's password stack to change a user's password. It also returns TRUE on success or FALSE on failure, and populates the optional error

parameter with an error message.

The PAM extension can be downloaded from its project page on the PECL site and compiled manually, or you can install it automatically by executing the command:

```
pear install pecl/pam
```

This command invokes the PEAR installer, which will download and compile the latest stable release. To compile the extension, you will need to install the libpam and libpam-devel packages available with most Linux distributions. The finer points of package installation vary according the distribution and you should consult your manual for details, but some common methods involve using the up2date, urpmi, apt-get or emerge tools.

The next step is to tell PHP to load the newly created extension via *php.ini*:

```
extension = pam.so
pam.servicename = php
```

Which configuration file in /etc/pam.d/ does your PAMenabled application use? As you've probably figured out by now, this will be determined by the service name, which in turn must be passed to PAM when invoking its API. pecl/pam uses the pam.servicename INI setting to determine the service name to pass along and sets the default value to php, so the default configuration file would be /etc/pam.d/php. This INI directive can be changed from all scopes—so it may be set in php. ini, httpd.conf, .htaccess or even from within your script using ini_set(). While this allows each application to have its own separate configuration and restrictions if needed, most setups should be fine using the default setting.

Listing 2 shows a simple configuration file for use with

In LESS Than ONE MINUTE You Can Run the Easiest, Best-Guaranteed, Re-brandable PHP Email Management Software On the Planet and Triple Your Email Marketing Success Rates!

You may have tried other email systems or auto-responder software, or even tried programming your own email software yourself. STOP!

This time, try the Rolls-Royce of PHP Email - oemPro:

- Manage newsletters with Ease
- Track click-throughs better, faster

User friendly

Send in any format

Discover how people just like you are producing high quality online newsletters, managing lists and maintaining customer information without all the headaches and hassles. Try it RISK FREE today, go to www.octeth.com



OemPro The International Rolls-Royce of PHP Email marketing software

a product of octob

PHP. It uses a fairly standard UNIX authentication setup, together with pam_warn.so for logging failures to syslog. The examples that follow next assume that the rules from Listing 2 are available in /etc/pam.d/php, or a path reflecting the name specified in pam.servicename. Note that the file must not contain any DOS linefeeds, or PAM might behave erratically.

Accessing Shadow

The standard UNIX authentication scheme uses a special file, /etc/shadow, to save the hashed passwords, and the PAM module must therefore be able to access that file. When running PHP as a SAPI module or from CGI, it is executed as your server's user and group. By default however, most Linux and *NIX systems are configured to only allow the root user to access the shadow file. One way around this might be to change permissions on the shadow file to group-readable and chgrp the file to a group the Web server user belongs to. However, before leaping into this, you should give some serious thought to the fact that allowing your Web server to read shadow—which contains all the password hashes, including the root password—might be a big concern security-wise. A more elegant solution is to use a privileged script that can act as an authentication proxy between the Web server and the rest of the system. Listing 3 shows the implementation of such a script. It reads parameters from STDIN and writes the result status to STDOUT, and—since it's a rather simple script—we can scrutinize it for security holes and eventually trust it to run as root. You can test it out by logging on as root, saving it to pam_proxy.php and running it from the command-line: php pam_proxy.php. To verify that your pecl/pam, php.ini and /etc/pam.d/php setup works you would type something like:

```
root@firefly tmp# php pam_proxy.php
auth
mikl
secretpassword
200 OK
```

In this example the authentication succeeded, since I provided my username mikl and my password. Had authentication failed, an error code plus an additional error message would have been printed instead of 200 OK.

The script also supports changing a password using the chpass command:

```
root@firefly tmp# php pam_proxy.php
chpass
mikl
secretpassword
evenmoresecretpassword
200 oK
```

In order to run this script from the Web server but still

have it execute as root, we'll use a bit of sudo magic. As root, execute the command visudo and add a line saying:

```
apache ALL = NOPASSWD: /usr/bin/php /path/to/pam_
proxy.php
```

This line states that the Web server user apache should be allowed to execute the command /usr/bin/php /path/to/pam_proxy.php as root without requiring a password. You might have to change apache to the user your Web server runs as, and you might also need to change the paths to match those of your php interpreter and pam_proxy.php script.

Now we just need some way of executing and interacting with the proxy from within PHP, and this is what the code in Listing 4 does by wrapping all the process handling in a handy utility function. It works by executing the proxy script with root privileges using sudo, writing the \$input parameter to the STDIN pipe \$pipes['0'] and then read the resulting status line from the STDOUT pipe \$pipes['1']. Before using this function, remember to adjust PAM_PROXY_PATH to point to where your sudo and php executables as well as the pam_proxy.php script have been installed. sudo is very picky about the commands it will execute, so take care to get the paths exactly the same as you typed them in visudo.

Listings 5 and 6 show some examples of using the util-

LISTING 5

```
<?php
   require_once "./listing3.php";
  4 if (isset($_REQUEST["username"], $_REQUEST["password"])) {
       $input = "auth\n"
       $input = strtr($_REQUEST["username"], "\r\n", " ")."\n";
$input .= strtr($_REQUEST["password"], "\r\n", " ")."\n";
      $valid = pam proxv($input, $error):
10 else {
       $valid = null;
12 }
14 ?>
15 <html>
      <h1>Login</h1>
       <? if (null !== $valid) { ?>
           <? if ($valid) { ?:
               <h2>Success</h2>
               Authenticated user <?= $_REQUEST["username"] ?>
           <? } else { ?>
               <h2>Failure</h2>
               <?= $error ?>
           <? } ?>
       <? } ?>
26
27
       <form method="post">
           28
                   Username:
                   <input type="text" name="username" value="<?=
                       htmlspecialchars($_REQUEST["username"]) ?>" />
               Password:
                   <input type="password" name="password" />
               <input type="submit" value="Login" />
39
       </form>
40 </html>
```

Online Training Courses from php|architect

Zend PHP Essentials

Our introductory PHP course, *Zend PHP Essentials*, was developed for us and Zend Technologies by PHP expert Chris Shiflett, co-founder of the PHP Security Consortium. This **19-hour course** provides a thorough introduction to PHP development, with particular care to "doing things right" by covering **security**, **performance** and the best development techniques. Rather than cramming as much theory as possible, PHP Essentials provides a thoroughly practical approach to learning PHP—thus ensuring that each student will be able to write good PHP code in a **real-world setting** by the end of the course.

Zend PHP Certification Training

If you want to become a Zend Certified Engineer, this course is the best preparation tool that you'll ever find! Designed by some of the same Subject Matter Experts who also helped write the exam itself, this course covers every single topic that is part of the exam. The Zend PHP Certification Training (course) provides a complete overview of the exam, and doubles as an excellent refresher course in PHP for any developer.

Zend Professional PHP Development

This is our advanced course for the professional PHP developer. This course picks up from where *PHP Essentials* ends and provides a thorough, indepth analysis of advanced features found in both PHP 4 and PHP 5, including object-oriented programming and design patterns, XML development, regular expressions, encryption, e-mail manipulation, performance management and advanced databases.

Course	Description	Start Dates	Duration	Tutoring	Prerequisites	Cost
Zend PHP Essentials	Covers PHP 4 and PHP 5 Provides a thorough practical introduction to PHP Covers security and performance	Anytime	7 Sessions 19 Hours 3 Weeks	YES	-	\$769.99 US (\$999.99 CAD)
Zend PHP Certification Training	Covers every topic in the exam Provides an excellent refresher course for PHP at all levels	Anytime	7 Sessions 19 Hours 3 Weeks	YES	Zend PHP Essentials	\$644.99 US (\$838.99 CAD)
Zend Professional PHP Development	Covers advanced PHP 4 and PHP 5 topics Perfect for going "beyond the basics" and learning the true power of PHP	Anytime	7 Sessions 19 Hours 3 Weeks	YES	Zend PHP Essentials	\$769.99 US (\$999.99 CAD)

- All our courses are delivered **entirely online** using an innovative system that combines the convenience of the Internet with the unique experience of being in a real classroom.
- All sessions take place in real time, and the students can interact directly with the instructor as if they were in a real classroom either via voice or text messaging.
- In most cases, our system requires no software installation and works with the majority of operating systems and browsers, including Windows, Mac OS and Linux, as well as Internet Explorer, Firefox and Safari.
- All courses include a generous amount of homework and in-class exercises to ensure that the students assimilate each topics thoroughly.
- Tutoring is available (via e-mail) throughout the duration of the entire course.
- Each class includes a complete set of recordings that the students can peruse at their leisure.



LISTING 6

```
require_once "./listing3.php";
       $_REQUEST["username"], $_REQUEST["oldpass"]
       $_REQUEST["newpass"], $_REQUEST["newpass2"])) {
       $valid = false:
       if ($_REQUEST["newpass"] != $_REQUEST["newpass2"]) {
            $error = "New passwords does not match":
12
13
14
15
16
       else {
           $input = "chpass\n":
           $input = strtr($_REQUEST["username"], "\r\n", " ")."\n";
$input := strtr($_REQUEST["oldpass"], "\r\n", " ")."\n";
$input := strtr($_REQUEST["newpass"], "\r\n", " ")."\n";
18
           $valid = pam_proxy($input, $error);
19
20 }
21 else {
       $valid = null;
23 }
24
25 ?>
26 <html>
       <h1>Change password</h1>
       <? if (null !== $valid) { ?>
          <? if ($valid) { ?>
29
30
               <h2>Success</h2>
               Changed password for user <?= $_REQUEST["username"] ?>
           <? } else {
               <h2>Failure</h2>
               <?= $error ?>
35
36
           <? } ?>
       <? } ?>
       <form method="post">
38
           39
40
                   Username:
                   <input type="text" name="username" value="<?=
42
                       htmlspecialchars($_REQUEST["username"]) ?>" />
43
               45
                   >01d password:
                   <input type="password" name="oldpass" />
46
47
               49
                   New password:
50
                   <input type="password" name="newpass" />
               New password (repeat):
54
                   <input type="password" name="newpass2" />
               <input type="submit" value="Change password" />
       </form>
59 </html>
```

LISTING 7

```
# This stack is called when authenticating a user
         required pam_nologin.so
auth
          sufficient pam_unix.so likeauth nodelay
auth
auth
         sufficient pam_krb5.so use_first_pass no_user_check
realm=example.com
         sufficient pam_ldap.so use_first_pass
auth
         required
auth
                     pam_warn.so
                     pam_deny.so
          required
# This stack would be called when changing a password
password sufficient pam_unix.so md5 shadow nodelay
password sufficient pam krb5.so use authtok no user check realm=example.
com
password sufficient pam_ldap.so use_authtok
password required
                     pam_warn.so
password required
                     pam_deny.so
# This stack is not relevant in most web-server setups
account required
                     pam permit.so
session
         required
                     pam_permit.so
```

ity function pam_proxy() and the helper script in order to perform authentication and password changes. Basically the pages are simple HTML forms that post to themselves and use the proxy function to interact with PAM. If you follow the instructions concerning sudo, save the pages to your Web server as *listing{4,5,6}.php* and load them in a Web browser, you should be able to verify that basic authentication works. If all is well, you could even try changing your password using the script in Listing 6. Should something go wrong, the Apache error_log and the various system logs in /var/log/ might provide some clues, and don't forget to include the pam_proxy() function from Listing 4.

Sounds like an awful lot of trouble to go to just to authenticate a user? Remember that the approach with sudo and a proxy script is only really necessary when using an authentication scheme that needs root privileges. Most other schemes—Kerberos, NIS, LDAP— have no such requirements, and will quite happily run as the Web server user. In this case you can use the PAM functions directly from the Web application:

```
$username = $_REQUEST['username'];
$password = $_REQUEST['password'];
$valid = pam_auth($username, $password);
```

Advanced Setup

Let's move on to a more advanced example. Listing 7 shows an example configuration file for use with PHP, and as you can see it specifies that UNIX, Kerberos and LDAP authentication should all be attempted before PAM finally gives up. If you compare this example file with others in /etc/pam.d/ you may notice some differences, for example that the account and session groups are empty except for pam_permit.so and the additional no_user_check parameter. Without these rules and the extra parameter, all users would need to be present on the local system, for example in /etc/shadow or through NIS. This is certainly not the case in most Web server environments, where only the administrator has shell access. The additional realm argument passed to pam_krb5. so states that the default Kerberos "realm" or domain should be example.com. This would allow Kerberos users to authenticate without specifying the domain part of their account names, e.g. mikl rather than the full mikl@ example.com. It might be a good idea to include this parameter, considering that very few users would be likely to know what a realm is. This configuration also has the benefit that the end user doesn't need to know beforehand which specific authentication scheme/username combination to try, but can let any of their configured pairs have a go at it.

There is also the additional module *pam_nologin.so*, which you might have come across when looking at /

etc/pam.d/login or /etc/pam.d/system-auth. It disallows logins based on the existence of the /etc/nologin file; if it is present, only root is allowed to log in. This is mostly used to restrict logins while the system is booting and before it is fully configured and running.

This configuration would not be enough to satisfy the needs of <code>pam_krb5.so</code> and <code>pam_ldap.so</code>, both of which will need to know further configuration details (for example, the name of the server to connect to). Such module-specific configuration data is stored separately in <code>/etc/krb5.conf</code> or <code>/etc/ldap.conf</code>, and conforms to the given module's own configuration format.

Summary

When faced with the problem of supporting multiple and changing authentication schemes, it makes sense to look for a pre-existing solution. Not only does such a solution free you from having to build support for every little authentication scheme into your application, but it also offers a flexible and modular authentication framework that can be customized to suit the customer's needs. Coupled with an authentication scheme such as Kerberos, NIS or Active Directory, PAM is a real breeze to use. If you're relying on UNIX authentication in an HTTPD environment things are a bit trickier but, as I've demon-

strated here, there are ways around those problems. In fact, in most cases, you would only use shadow authentication for testing and administration purposes. Most production servers I've seen use other forms of authentication, since manually keeping a shadow file in sync across multiple servers would be rather cumbersome.

When looking around for a pre-existing authentication framework you may encounter many options, but PAM is one of the simplest and most widely adopted solutions that fits the description. Where cross platform support is not an issue, PAM might well be the natural choice.

MIKAEL JOHANSSON has been involved with PHP for about four years now, and has authored or co-authored several PHP extensions. He makes his living developing enterprise applications using PHP, C# or Python. You can contact him at mikl@php.net or visit his website at http://www.synd.info/.



MySQL Babel

The Lord said: "If as one people speaking the same language they have begun to do this, then nothing they plan to do will be impossible for them. Come, let us go down and confuse their language so they will not understand each other."

Genesis 11:6

by Alessandro Rosa

Meet MySQL Babel. The project, currently at prototype stage, aims to open up PHP applications to user-defined database interrogation by translating native language sentences into SQL queries. The intended audience for this article is intermediate to advanced level programmers—and advanced database frontend users—who might be interested in helping develop the Babel project, and ultimately in using it.

QL (Structured Query Language) is the most widely used syntax for accessing databases all over the world and, on the Internet at least, often supported by a comfortable front-end interface written in PHP. After coding a number of Web based applications that were endowed with such database front-ends, I imagined a further step. It should be possible to broaden the range of user access options on offer by allowing user-defined queries on the fly in a high-level language, such as English.

This might sound odd to any developers reading this. It goes way beyond the relative strictness of database administration front-ends, where predefined query models set in stone by the developer are applied by users who have never mastered—or wished to master—SQL syntax. It is in this sense that the intended readership for this article includes experienced users of database

PHP: 4.4 or better

0/S: Any supported by PHP

Other Software: MySQL

Useful/Related Links:

http://freshmeat.net/projects/mysql_babel/

TO DISCUSS THIS ARTICLE VISIT:

http://forum.phparch.com/364

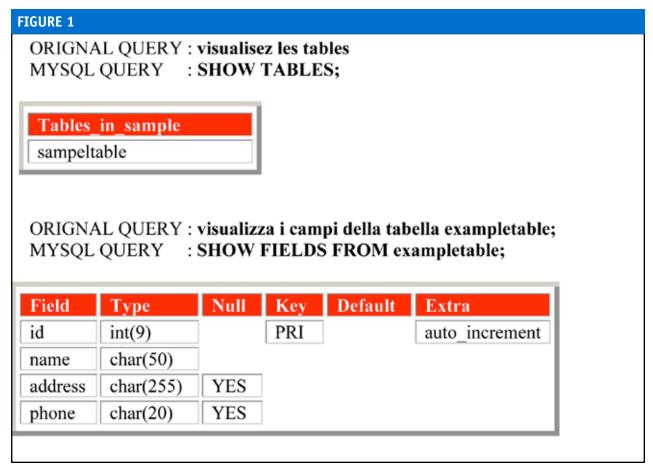
front-ends, as well as PHP developers capable of implementing classes or enhancing various aspects of this project. I believe that, once the concept behind my approach has been fully understood, the project itself will begin to fall into place.

The basic premise is that opening up a database-driven application to the widest range of versatility would allow that application to be more responsive to the needs of the end user. How often have you needed to run a one-off, a single special query that hasn't been anticipated or templated by the application developer? The MySQL Babel vision is all about helping the end user towards more freedom in his or her relationship with stored data, and ultimately about improving human-machine interaction.

The concept

There are elements of my original idea that bear closer investigation. For example, a user's language skills might be expected to deteriorate in a stressful situation, when they are probably going to be typing in a hurry. We also have to wonder how well English—usually seen as the ultimate high-level language—is generally mastered by non-native speakers. Perhaps we should go further,

then, and aim to support the needs of any user sitting in front of the screen, regardless of their level of skill in either SQL syntax or the English tongue. The operative assumption here is that the native language of the user represents the easiest way for that user to express himself. This assumption lies behind the concept of the MySQL Babel class, but the assumption itself still needs to be sharpened and given some sense of direction before it can become a driving principle. The chief goal of MySQL Babel re-stated, therefore, is to knock down the barriers to entry: first to knock down the barrier of the need for SQL syntax, and then to knock down the English language barrier. However, it needs to be recognised that knowledge of the basic structure of SQL architecture is the exclusive province of expert users and developers, and this in itself is a limiting factor. It limits us to providing a solution that will run queries originally formed in native languages. We can't ask the database something it doesn't understand; the user will still need to know that there is such a thing as a database, that it contains one or more tables, and that these tables contain information organised in blocks called 'fields'. That said, allowing queries in a less formal language than SQL should still be enough to bring down the entry level dramatically when it comes to database administration.



Take a look at the pair of screenshots displayed in Figure 1. These are pulled from a very basic example demonstrating the MySQL Babel prototype in action. It doesn't take a great leap of the imagination to envisage something like this usefully supporting the administration interface of a Web based application.

Translation or conversion?

Strictly speaking, MySQL Babel class tasks do not really include translation. Native input sentences are not parsed and interpreted here. Rather than true translation, it might be better to state that a conversion is performed: queries are simply turned into a different form, with no concern either for grammar or for sense beyond the structure of the database itself. That said, I will keep on using the word *translation*, because that seems to me the most apt term to describe the situation where information flows between different languages. The more pedantic among you will simply need to remember to apply the term in the same sense it is used here. Having simplified and reduced the meaning of translation makes it possible to approach the task in a very simple way in the MySQL Babel project. Why? Because an overview of

the target dialect, SQL grammar, shows that it is much simpler than any spoken language. There are no idiomatic forms, and there are very few grammatical rules in the language. Often an SQL query consists of a single command followed by a number of parameters, such as fields, table names or sorting criteria. In such cases, it doesn't take a huge effort to recover the sense of the original query, regardless of the language in which the request was made. The move from a complex to a simple grammar environment implies that the problem of transformation from native languages to SQL queries can be resolved in a relatively straightforward way.

As we will see later, many input expressions simply need to be sifted for chaff before they are finally turned into the form of an SQL query. This can definitely be viewed as an over-simplistic approach, but bear in mind that translation is a less than accurate science, and that even mainstream translation software can return extremely rough results. Adhering to the KISS principle ('Keep It Simple, Stupid') is one way of minimising the likelihood of failure, and is probably even more crucial in this context than it is in most software projects.

Translation

In Figure 1, you saw how Babel managed two native language queries about table and field content in an address book in French and Italian, respectively. Translation issues are quite easily resolved for such simple queries, and explanations are needless. So, before getting into MySQL Babel class internals, let's look at some slightly more complex examples, again revolving around the management of an address book. All our examples deal with a table named sampletable that contains just four fields: id, name, address and phone. Translation consists of three main stages. In the first stage, we want to set up the native language for Babel to work with. The options available at present include English, French, German and Italian. Here's an example of a French input query:

choisissez tous de sampletable

Tokenization is the name of the process that splits an input query into elements, each of which is separated by blank space. The example given here would produce four such elements:

choisissez tous de sampletable

Finally, each token is translated into SQL syntax, supported by the dictionary appropriate to the chosen language. In our initial example, every token in the French language query maps directly to a term belonging to SQL syntax—see Figure 2—making this one of the simplest possible queries from the perspective of translation.

Obviously, in more complex cases of queries this one-to-one correspondence will no longer hold. Let's take a look at the example in Figure 3, which demonstrates one of those slightly more complex queries. Here, we have the expression:

où id est plus grande que 3

...a seven token expression, which needs to be mapped to the four token expression:

WHERE id > 3

Somewhere along the way, we need to lose three tokens. In fact, verbose expressions like est plus grande que, typical of spoken languages, can easily be compacted.

php archite	Visit: http://more info	o://www.phparch.com/print for rmation or to subscribe online.		
php architect Subscription Dept. P.O. Box 54526 1771 Avenue Road Toronto, ON M5M 4N5 Canada	Your charge will appear under the name "Marco Tabini & Associates, Inc." Please allow up to 4 to 6 weeks for your subscription to be established and your first issue to be mailed to you. *US Pricing is approximate and for illustration purposes only.			
Name:	Choose a Subscription type:			
Address:	☐ Canada/USA	\$ 77.99 CAD (\$59.99 US*)		
City:	□ International Air□ Combo edition add-on	\$105.99 CAD (\$89.99 US*) \$ 14.00 CAD (\$10.00 US)		
State/Province:	(print + PDF edition)			
ZIP/Postal Code:				
Country:	Signature:	Date:		
Payment type:				
VISA Mastercard American Express □ □ □ □	*By signing this order form, you agree that we will charge your account in Canadian dollars for the "CAD" amounts indicated above. Because of fluctuations in the exchange rates, the actual amount charged in your currency on your credit card statement may vary slightly.			
Credit Card Number:				
Expiration Date:	To subscribe via snail mail - please detach/copy this form, fill it out and mail to			
E-mail address: Phone Number:		ve or fax to +1-416-630-5057		

Thus, in our example, only the term plus grande was taken as meaningful and turned into SQL syntax in the form of the 'greater than' operator >. The verbose terms est and que were completely ignored by the translation mechanism.

approach would be to have a common set of reference terms, namely, a kind of *dictionary*. From here, the process can be further divided into two types of dictionaries: *abstract* dictionaries, and *concrete* dictionaries. The *abstract dictionary* does not exist in terms of code. It is

I am actively looking for supporters to maintain a dictionary for their own native language.

It is important to understand that the core of the translation process relies largely upon a subjective relationship between native and SQL grammars. My simple direct mapping example was an exception among the far more common complex cases represented by Figure 3. Thinking about the way queries should be handled in general becomes slightly more difficult from here on in.

That said, the point of having these mapping examples is to make you aware of the wide range of native language terms that can be reduced to a simple SQL equivalent. The fundamental concept here is that every input token must be turned into another token. This is not as contradictory as it may at first seem, if we also allow that a token can consist of an empty string where there is no SQL correspondence.

Dictionaries

Obviously, successful translation relies upon the setting up of a common basis. One initial difficulty during the development of the MySQL Babel prototype was to find a basis to adopt. In the event, I decided that the best

choisissez tous de sampletable

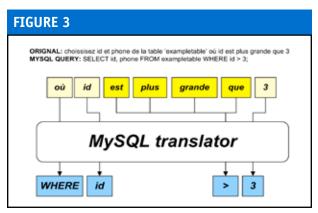
MySQL translator

SELECT * FROM sampletable

simply conceptual, and could be described as a set of directives to assist those implementing a new concrete native language dictionary. Linguistic analysis should provide much of that assistance. For example, conjunctions, such as and, with and or, should find equivalents in SQL, whereas articles—whether definite or indefinite—should be left out of the equation altogether and get the 'empty string' treatment.

Concrete dictionaries are structured in terms of code. They are manifested as bi-dimensional arrays, where each indexed entry includes both the native term and the related SQL index value. The SQL index value is the key shared by all concrete dictionaries. Refer to Figure 4 for an illustration that should help clarify this point.

The SQL indices codify the action to be performed. For example, the code 0002 tells us to ignore the input term during the translation process or, more accurately, to turn it into an empty string. Any other code tells us that the process should continue. Where the index matches, the class picks up the associated SQL term; if the search fails, and no matching term is found in the concrete dictionary, the original word is copied directly and appears



as part of the translated query.

Each concrete native language dictionary array is stored in a file named according to the ISO 639 language code, e.g. it for Italian, fr for French and en for English. The file includes a group of functions that will dynamically detect the version, the author and the name of the file, followed by a second group of functions—but more of those later.

As we have already seen, native language dictionaries need to include terms that have no SQL equivalent, both in order to eliminate them from the process and to allow something more nearly approaching fluency in native language queries. As you will see later, synonyms and idiomatic language usage are also provided for, with much the same aims in mind.

A basic implementation

Here is some much-simplified code written to demonstrate usage of the MySQL Babel prototype:

```
require('mysql_babel.php');
$lang = new languages($language);
if (!$lang->isClassOperative()) {
    echo $lang->getErrMsg();
    exit;
}
```

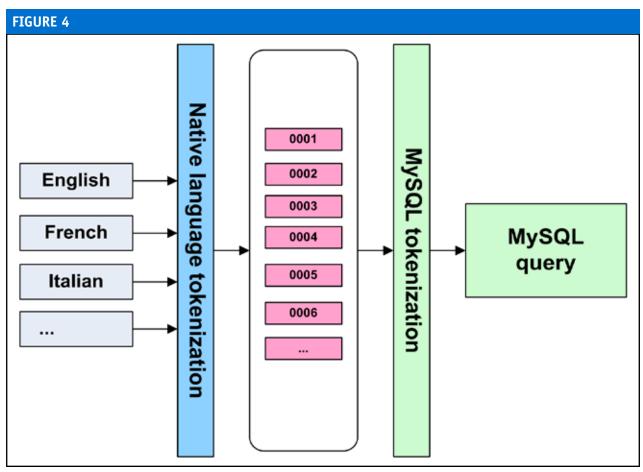
```
$tr = new mysql_babel($host, $db, $user, $pass);
$tr->insert_language_code($language);

$tr->native_dict_array = $lang->get_native_diction-
ary();
$tr->mysql_dictionary_array = $lang->get_mysql_dic-
tionary();
$tr->errors_messages_array = $lang->get_errors_ar-
ray();

$tr->insert_mysql_source_cmdline($query);
$tr->translate();
$tr->run_query(true);
```

Notice that the main class, mysql_babel, works alongside another class devoted to dictionary management, namely the languages class. The use of a different class to manage the dictionaries was a specific choice aimed to avoid resource overload; this 'slave' class is used and dropped as required by the 'master' mysql_babel class.

The database credentials should be set in a configuration file. Before the database credentials are input, the languages class method isClassOperative() is called to check that the language passed to it—which may be specified either in a configuration file or in the user interface—is actually supported by an appropriate dictionary file. If that test fails, the code displays an error message and exits. Assuming the chosen language is supported, the database connection is made, the



mysql_babel class is instantiated, and two dictionaries are loaded, one for the native language, and the other for SQL syntax. This is achieved by having the class pick up three (not two!) arrays:

```
$tr->native_dictionary_array=$lang->get_native_dic-
tionary();
$tr->mysql_dictionary_array=$lang->get_mysql_dic-
tionary();
$tr->errors_messages_array=$lang->get_errors_array();
```

and store them in their analogous containers within mysql_babel. The first array contains terms from the se-

It is important to note that a working SQL query is more likely to result from a correct native language sentence, that is, one written with some respect for grammar.

This early version of mysql_class both runs the query and displays the output as part of the same method, run_query; the Boolean parameter passed to it switches the display on or off. I would agree that it isn't very elegant to have a single member function achieve these very different tasks, and future revision is likely to split the functionality across two separate methods. Still, here is the code currently used to display field names as table headers:

The chief goal of MySQL Babel is to knock down the barriers to entry.

lected native language; the second array comprises the matching list of SQL commands; and the third array contains a list of error messages, which are also provided in the language dictionary file. All three arrays are dynamically loaded by mysql_babel member functions. Finally, the native language input query is picked up from, say, a text box or other element of the application interface. Having been given the appropriate validation checks (this being currently the responsibility of the application), it resides in the variable \$query. We then plug \$query into the mysql_babel class:

```
$tr->insert_mysql_source_cmdline($query);
and call the translate() method:
$tr->translate();
```

translate() runs through a few subroutines to further test input and output data integrity; this needs building upon to remove the onus on data validation from the application developer.

Finally, we run the resulting SQL query:

```
$tr->run_query(true);
```

and the database response should be displayed as a table, assuming the query was semantically correct, i.e. all the field and table names used were right and the original language structure made sense to the database.

```
echo "\n";
while ($cols = mysql_fetch_field($h_qry))
{
  echo "";
  if ($cols) echo "$cols->name ";
  echo "\n";
}
echo "\n";
```

and the rest of the output data, obviously, is per row.

Idiomatic expressions

The translation process includes three separate subprocesses designed to prevent potential errors during translation and refine the input query. First, besides the core language dictionary, each language file contains a list of idiomatic expressions. This has the potential to eliminate one of the translation difficulties outlined in the introductory sections. The code—basically, a series of string replacements—is run prior to the standard tokenization routine. For an example, here is part of the Spanish version:

```
function pre_idiomatic($strIn) {
    $strOut = str_replace("todos campos", "todo",
$strIn);
    $strOut = str_replace("todos los campos",
"todo", $strOut);
    $strOut = str_replace("cada campo", "todo",
```

```
$strOut);
    return $strOut;
}
```

The pre_idiomatic() member function fits the input requirements of the translation kernel, producing single tokens that will later be turned into single SQL terms. An example follows, listing the same phrase written in the currently available native dictionaries:

- FRENCH: choisir chaque champ de sampletable;
- GERMAN: wählen alle Felder von sampletable;
- ENGLISH: select *each field* from sampletable;
- ITALIAN : seleziona *tutti i campi* dalla tabella sampletable;
- SPANISH: seleccionar todos los campos desde la tabla sampletable;

The idiomatic expression (in italic script) initially becomes the local equivalent of all, and the tokenizer then translates all to an asterisk (*). Thus, all the above queries will eventually become:

```
SELECT * FROM sampletable;
```

pre_idiomatic() is a many-to-one function, in that a group of terms, often variations of a single idiomatic form, will be reduced to a single native language token. This is a way of ensuring that the input query has a good chance of meeting the standard required by the tokenizer, and consequently by SQL.

Safety checks

There is the possibility that a native query might include quoted text that correlates with SQL reserved words, which would cause errors during tokenization and thus the translation process. The member function safe_data() was coded in an effort to avoid this. It is called twice from within the translate() method: once before the translation, when string protection is locked, and again when the process is over, to unlock string protection. The function takes two arguments, the input query \$qry_input and the lock action \$bLock (a Boolean value).In this example, the input data is written in Italian, but it could equally well be any other supported language:

```
inserisci nella tabella sampletable i valori
(24,'Jackob', 'Contrada per caso','344-4543876');
```

Here, the quoted string Contrada per caso contains the word per, which would normally be tokenized as 0039 and rendered as BY in SQL syntax. The safe_data() function protects quoted strings from translation by replacing any whitespace with the string @@. This ensures that

the generated tokens are not words in any language, and so will be rendered as literals. Because safe_data() is called at an early stage, what is passed to the tokenizer looks like this:

```
inserisci nella tabella sampletable i valori
(24,'Jackob', 'Contrada@@per@@caso','344-4543876');
```

which in turn becomes:

```
INSERT INTO sampletable VALUES
(24,'Jackob','Contrada@@per@@caso', '344-4543876');
```

The procedure is of course reversible. Calling safe_data() a second time, this time with the lock set to FALSE, removes all string pads and recovers the original form. Finally, we have the correct MySQL query:

```
INSERT INTO sampletable VALUES
(24,'Jackob','Contrada per caso','344-4543876');
```

Without any such input data protection, the final output would have been:

```
INSERT INTO sampletable VALUES
(24,'Jackob','Contrada BY caso','344-4543876');
```

because the content of the quoted string would also be affected during tokenization. There may, of course, be better ways to offer such protection for quoted strings; nothing here is set in stone.

Finalization

Before the SQL query is returned from the translate() method, it's a good idea to check that it makes sense before the user can be allowed to run it. A translated query at this stage might conceivably include redundant expressions, such as SELECT TABLE or SELECT FIELDS where there should be only the SELECT command. Or it may contain an illegal sequence of operators, such as

```
< , =
```

with whitespace between each item. Such a sequence would need to be turned into the correct form of the less than/equal to operator <=.

Finalization therefore plays a very relevant role in improving the chance of a native language query's being rendered as legal SQL syntax. This is not absolutely error-proof at this stage. Incorrect queries may be retrieved, but any effort that is put into improving either the pre_idiomatic() or finalize() methods has an immediate and obvious impact on the rate of errors found in the final SQL query. The more the code is improved, the better MySQL Babel works; the efficiency of the class is entirely in the hands of the developers that manage those two methods.

The best way to optimize those development efforts

would be by having developers work as an open source team. Outside of anything else, the accuracy of the pre_idiomatic() and finalize() methods will increase as the number of errors discovered in the existing versions grows. And, particularly when it comes to pre_idiomatic(), the accuracy of the existing version is limited to the author's knowledge of a range of foreign languages; native speakers, whether programmers or not, would definitely be able to improve upon this!

Call for developers

The prototype for the MySQL Babel project can be downloaded from http://freshmeat.net/projects/mysql babel/. Note that the status of the MySQL Babel project at this stage of development is primitive, especially considered in the light of the manifesto given below. You have only to look at the language dictionaries to understand that they don't pretend to be absolutely correct, or even functional, at present. They have only been included in the current project download to set the stage for future development and to provide some basic examples of how the class might work with different languages. Given that I don't happen to have linguistic competence across a wide range of languages, I am actively looking for supporters who will take on the responsibility of maintaining a correct and full version of their own native language dictionary. There are several potential development paths for the project. For example, one topic open to discussion would be whether MySQL Babel could be made to work with native languages whose sentence syntax do not follow the sequence

verb -> object -> complements

The prototype implementation of MySQL Babel was achieved with languages that use this syntax (Italian, French, English and Spanish). It is likely to show up as a restriction, and probably the core code should be revisited to allow the class to work with languages that have a different grammatical structure.

sentence MySQL Translator MySQL query native errors dictionary Languages

Manifesto

Even with contributors able and willing to maintain the language files, there would still be far too much work for one person to take on alone. The full development of the MySQL Babel project would of necessity be a much a larger project, consisting of six main areas: - Directory development and expansion

- Core class improvements
- Finalization improvements
- Documentation
- Extension to other database engines
- Porting to other programming languages

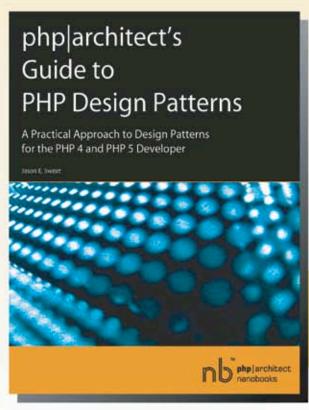
Clearly, the first point needs the support of contributors with a wide range of native languages, and an international task force is envisaged here. Such contributors would not necessarily need to be developers, although development skills will obviously be needed when it comes to improving the parsing of native language sentences. Documentation, similarly, would need the support both of developers (to generate skeleton manuals on a per-language basis from the source) and of native language speakers who will not necessarily have programming skills.

Programming skills are essential when it comes to the third and fourth points, and familiarity with SQL would obviously be a huge bonus there too. Across all areas, though, anyone interested in the project could usefully test the class in action and report the bugs they find in their native language implementation.

The final two points in my manifesto open up wider possibilities. MySQL Babel is, or will be, the PHP implementation of a concept that could equally well be coded into different programming languages or adapted to operate with database management systems other than MySQL. In this way, MySQL Babel might become part of a wider project, encompassing a range of SQL architectures, and more simply defined as SQL Babel.

ALESSANDRO ROSA is an Italian freelance programmer who has been working with Web and stand-alone applications for over a decade. His other interests include research into mathematical programming, analysis and history. Alessandro can be contacted through his home page at http://www.malilla.supereva.it.

ESSENTIAL BOOKS from php | architect



Over 16 Design Patterns presented the way it matters to PHP developers!

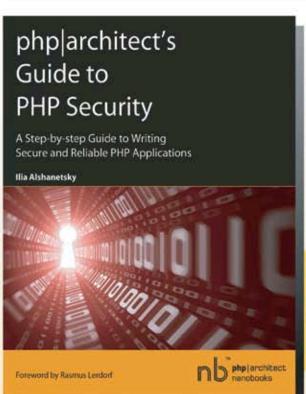
You have probably heard a lot about Design Petterns—a technique that helps you design rock-solid solutions to practical problems that programmers everywhere encounter in their day-to-day work.

Even though there has been a lot of buzz, no-one has yet come up with a comprehensive resource on design patterns for PHP developers---until today.

Author Jason E. Sweat's book php|architect's Guide to PHP Design Patterns is the first, comprehensive guide to design patterns designed specifically for the PHP developer. This book includes coverage of 16 design patterns with a specific eye to their applications in PHP when building complex web applications, both in PHP 4 and PHP 5 (where appropriate, sample code for both versions of the language is provided).

- Includes over 16 design patterns
- Each patter is discussed in detail with practical code applications
- Covers both PHP 4 and PHP 5
- Provides a thorough test-driven approach to design patterns
- All source code available online

For more information, visit http://www.phparch.com/gpdp



Step-by-step Guide to Security: a Must-Have Reference Book for every PHP developer!

Security is a hot topic these days, with new exploits and patches released on a daily basis for all sorts of operating systems and applications. Recently, the security bubble has expanded to touch the PHP world, and several well-known applications have been the target of a great number of attacks.

Despite all the negative publicity, PHP is and remains a very stable--and very secure--programming environment. php|architect's Guide to PHP Security, written by security expert and frequent php|architect contributor Ilia Alshanetsky, provides you with a guide that covers everything you need to secure existing PHP applications and write new ones with security in mind.

- Provides techniques for both PHP 4 and PHP 5
- Includes a step-by-step guide to securing your applications
- Provides comprehensive coverage of security design
- Teaches you how to defend yourself from hackers
- Shows you how to distract hackers with a "tar pit" to help you fend off potential attacks

For more information, visit http://www.phparch.com/pgps

FileMaker for PHP Developers

FileMaker is a popular and powerful desktop database application toolkit. FileMaker, Inc. recently released a beta version of the FileMaker API for PHP, which allows PHP to more easily talk to the FileMaker Server Advanced product. Last month, author Jonathan Stark introduced some of the concepts behind the newly hatched API. In the concluding episode of this two-part series, he explains how FileMaker makes editing your database records a snap.

by Jonathan Stark

PHP: 4.3.x or better

0/S: Any supported by PHP

Other Software: FileMaker Pro and FileMaker Server Advanced

TO DISCUSS THIS ARTICLE VISIT:

http://forum.phparch.com/365

was designed to allow knowledge workers to quickly and easily construct data management systems for themselves.

In the first part of the FileMaker for PHP Developers series, I introduced you to the basics of FileMaker development in the desktop environment, and explained how to leverage that development work to easily display your data on the Web using the FileMaker API for PHP. I covered the terminology used by the FileMaker API, introduced the list view and form view layouts, and explained how having the business logic embedded in the layout can be a surprisingly efficient approach in real-world applications. We then went on to look at views in more detail, and ended with a brief exploration of where (and why) FileMaker might be deployed to the greatest effect.

With much of the FileMaker desktop development basics behind us, we can focus more on the PHP side of things. This time around, you will learn two different techniques for updating your database records.

Updating a Single Record

Loyal php|architect readers will recall the <code>view_products.php</code> script, which was the first of the code listings from Part One of this series. If you recall, we used this script to display a searchable and sortable list of products from the <code>ProductCatalog</code> database, which is included with the <code>FileMaker API</code> for PHP download bundle in the form of <code>ProductCatalog.fp7.To</code> make the edit functions accessible from the demo scripts available to you, I have simply changed the <code>view</code> link beside each product listed on that page to an edit link by altering the line:

```
$page_content .= '<a
href="view_product.php?recid='.$record_object-
>getRecordId().'">view</a>';
```

to read:

```
$page_content .= '<a
href="edit_product.php?recid='.$record_object-
>getRecordId().'">edit</a>';
```

This new link will navigate to a form view of the clicked product. However, before we can look at the PHP code used to update the selected product record, we really need to take a peek at the corresponding layout in File-Maker Pro.

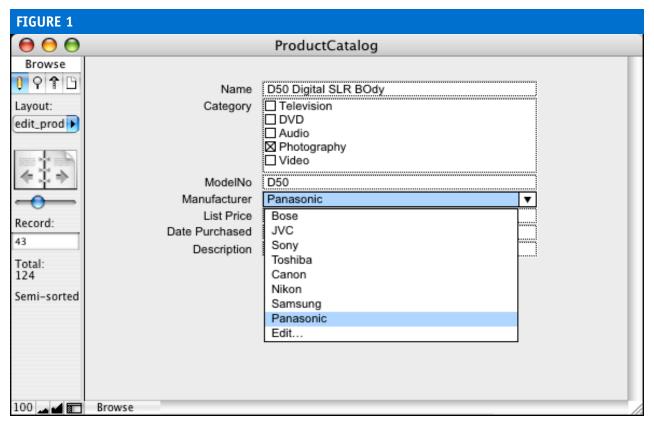
Figure 1 shows the FileMaker layout in Browse mode, which is also known as data entry mode. Notice that there are radio buttons applied to the Category field here, and there is also a drop-down list attached to the Manufacturer field. It's obvious that editing these value lists will be a trivial task, even for novice FileMaker Pro users. The end user need only click the Edit... link at the bottom of the list to be presented with the Edit Value List dialog shown in Figure 2. You will see in a moment that this is very cool, because this simple action on the part of the user will trickle through to the Web without any changes being made to the PHP code.

The code that makes up edit_product.php is reproduced in full in Listing 1. For the sake of clarity, I have

left out large chunks of form validation and the sanitization of user input. These are important concerns and relevant here, but a discussion of general form submission handling is outside the scope of this article. It is also notable that I left out much of the FileMaker error checking because it is very repetitive and does not serve to illustrate my point.

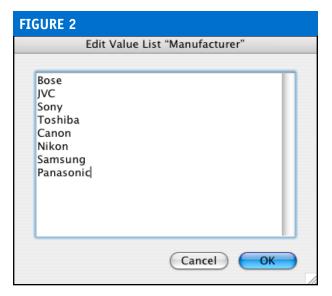
You will notice that the script is divided into four distinct sections. The first of these, *initialization*, opens with the definitions used for database connection. For reasons of security, any database credentials should be stored in a separate configuration file above the document root and included from there. Other items in the script initialization section include the require_once() call representing the FileMaker dependency, global variable initialization, and some code to check the status of the \$_POST array to determine whether the form has yet to be processed. If the process_form element has been set, the result of the form processing will be displayed in the browser above the empty form; otherwise, the empty form alone will be displayed.

The second section is all about form display. It contains the function show_form(), which takes all its cues from the specified FileMaker layout. The fields that have value lists applied in FileMaker will be formatted appropriately in HTML, depending on the style type associated with the underlying field object. Note that everything here is completely dynamic, so that changes made to the



FileMaker layout or to the values lists on that layout will be reflected in the HTML page without any modification of the PHP code.

Thirdly, there is the *form processing*, which takes place, unsurprisingly, within the process_form() function. As with the show_form() function, process_form() bases all its logic on the FileMaker layout named at the beginning of the function; in this case, the chosen layout is edit_product. When the time comes for the record to

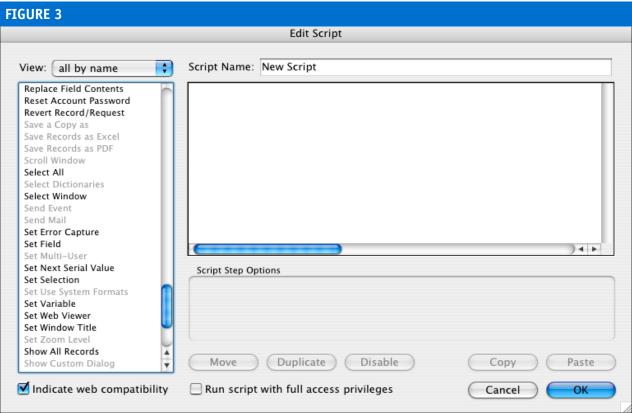


be updated, PHP queries the layout object for the fields it contains, using \$layout->getFields(). It then loops through the array of fields and matches the field names with those in the \$_REQUEST superglobal array. On finding a match, it pulls the corresponding data out of the \$_REQUEST array and updates the field value. Finally, it submits the change to the database. It is important you should be aware that there is a *lot* of validation missing from this area in particular, as mentioned earlier; a database should **never** be updated with raw user input in any real-life application.

With that out of the way, the final section of the script is dedicated to *HTML rendering*. Since this is a demo script, I chose to have the CSS style definitions inline rather than force an unnecessary listing upon you. Apart from that and the title, all we have here is a back link to *view_products.php* and the HTML content generated by show_form() and process_form(), if applicable.

Updating a Group of Records

Technically, it would be possible to update a group of records by simply expanding on the "single record update" technique, feeding the script an array of record IDs in a do.. while loop. However, this would be less than optimal from the performance perspective, since a) it would require a call to the server for every single record and b) the data is transmitted as XML. A better option would



be to use PHP to call a FileMaker script that will do all the dirty work for you; and that's precisely why there are FileMaker scripts.

FileMaker Scripts

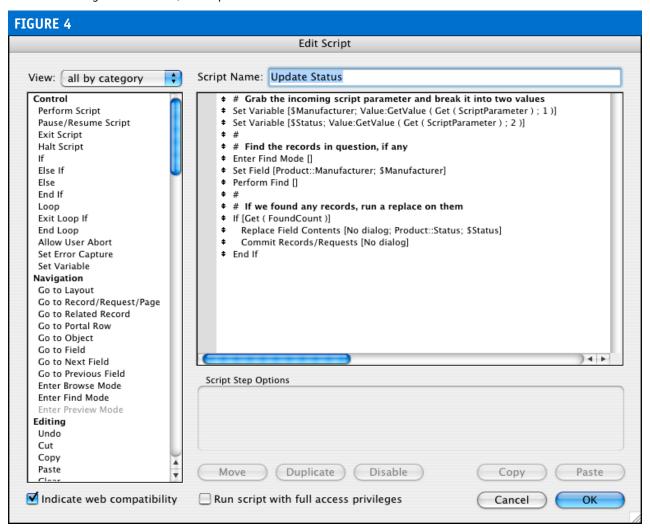
FileMaker Pro has a point-and-click scripting environment called *ScriptMaker*. This ScriptMaker allows you to create macros that can execute all sorts of useful commands with a great deal of ease. Normally, scripts are run by FileMaker Pro users, but they can be triggered by PHP as well. The coolest part is that you can send parameters to a FileMaker script via PHP, thereby customizing the behavior of that script on the fly. In this example, I am going to create a PHP page that will allow the user to select a Manufacturer, enter a Status and submit the form. The form will send the Manufacturer Name and Status to the Update Status script in FileMaker, passing all the data elements as arguments. The FileMaker script will then locate any Product records with a matching Manufacturer, and update the Status

value accordingly.

Figure 3 is an illustration of the ScriptMaker environment in FileMaker Pro. Hopefully you can see from the image that it's quite simple to use. The area on the left contains a list of the available commands, and you can double click on any of these to move them into the text area on the right, which displays the script itself. Not all the commands that are made available in ScriptMaker are compatible with PHP, so I have activated the Indicate web compatibility checkbox; those of the command options that can't be used are grayed out as a result.

Figure 4 is the Update Status script itself. As you can see, it is very short, and in fact it only took about three minutes to write. It would have taken me much longer to write it in PHP and, as I mentioned earlier, the performance obtained in this way would have been less than wonderful.

Let's break down that Update Status script and see what it's made of.



Update Status

The first section in the Update Status script accepts the incoming script parameter, breaks it into two values, and stores the values in separate variables:

```
Set Variable [
    $Manufacturer;
    Value:GetValue(Get(ScriptParameter); 1)
]

Set Variable [
    $Status;
    Value:GetValue(Get(ScriptParameter); 2)
]
```

Technically speaking, a FileMaker script can only accept one parameter, and you should access that parameter value with the Get(ScriptParameter) function. You can get around the single parameter limitation, as shown here, by delimiting your values with returns and using the GetValue() function. GetValue() accepts an EOL-delimited list of values and a value number as parameters, and will return the value indicated by the number. If you think of the EOL-delimited list as an array, then GetValue(\$Values; 2) is equivalent to \$Values['2'] in PHP.

Now that we have the number of arguments we need to pass, the next thing is to find the Product records that are associated with the selected Manufacturer name. We do this by entering Find mode, inserting the selected Manufacturer name into the Manufacturer field, and performing the Find request. While we're there, notice that the Product:: prefix in the Set Field step indicates that the Manufacturer field belongs to the Product table.

```
Enter Find Mode
Set Field [
   Product::Manufacturer;
   $Manufacturer
]
Perform Find
```

At this point, we need to check to see whether our request matched any records. To do so, we open an If block and make our enquiry using the function Get(FoundCount), which will return an integer. If the integer it returns happens to be 0, the If condition will evaluate to FALSE and the rest of the script will be skipped. If, however, the number of items is greater than 0, the If condition will evaluate to TRUE. This will trigger the execution of the Replace and Commit Records/Requests commands.

```
If [ Get(FoundCount) ]
  Replace Field Contents [
    Product::Status;
    Replace with calculation: $Status
]
  Commit Records/Requests
End If
```

The call to Replace does just as you might expect—name-

ly, it replaces the value in the Status field of the found Product records with the value in the \$Status variable. Remember this: the \$Status variable was populated by the script parameter that was sent from PHP.

When the Replace routine has completed, the Commit command is executed; this routine is responsible for writing the changes to the database.

update_status.php

With the FileMaker script in place, we can turn our attention to the PHP page that will call it: update_status.php, rendered here as Listing 2. As with the earlier code listing, I have left out much in the way of form validation and the sanitization of user input, so please tread with care when it comes to implementing this functionality yourself. There are five distinct sections in update_status.php, some of which match the sections in edit_product.php (Listing 1) and some of which are unique to this script. Thus, as before, we have the *initialization* stage making the decision about the nature of the HTML page content, depending on the stage of processing the script has reached. We meet, once again, the form display section containing the show_form() function, where the options for the select block in the Manufacturer field are pulled from the layout in FileMaker. Next up, there's something you haven't seen until now; form validation. In this instance, this is restricted to checking that the Manufacturer and Status fields contain some input, and ensuring that \$_POST['manufacturer'] doesn't contain an illegal hyphen or \$_POST['status'] any HTML tags. Again, this offers very little protection, and you will need tighter control over your user input data in any real-life application.

Next up is our old friend form processing, which is the home of the process_form() function. In this particular case, process_form() is the focus of the example, as it shows you how to go about sending a form submission to a FileMaker script. As an added bonus, you can see the syntax for sending multiple parameters in the line:

```
$script_param = $_POST['manufacturer']."\n".$_
POST['status']:
```

Remember, though, that "\n" is not valid syntax on every platform. Those of you who are running PHP 5.0.2 or newer will be able to use the built-in constant PHP_EOL here, but if you're stuck with an older version of PHP you will need to create your own EOL constant to achieve portability.

Finally, there is the HTML template, which once again contains inline CSS style definitions, a title, and the absolute basic necessities to frame and decorate this dynamically rendered page.

LISTING 1

```
2 /* edit_product.php */
  4 ########################
          INITIALIZATION
  6 #######################
  8 # For security reasons, these lines should either be included from a
  9 # config file above the document root, or possibly captured during a
 10 # login and stored in the SESSION superglobal array
12 define('FM_HOST', '127.0.0.1');
13 define('FM_FILE', 'ProductCatalog.fp7');
14 define('FM_USER', 'esmith');
15 define('FM_PASS', 'f!r3crack3r');
 17 # this is the include for the API for PHP
 18 require_once ('FileMaker.php');
20 # initialize page content var
21 $page_content = '';
 23 # if this page has been submitted to itself, then process it
24 if (array_key_exists('process_form', $_POST)) {
      $page_content .= process_form();
 26 }
 28 # show the form
 29 $page_content .= show_form();
 FORM DISPLAY
 32 #
 33 ##########################
 34
 35 function show_form() {
        # grab the record id sent in the url from list page or a post from
this page
        $recid = (array_key_exists('recid', $_REQUEST)) ?
htmlspecialchars($_REQUEST['recid']) : '';
        # set the layout name for this page
$layout_name = 'edit_product';
 42
         # initialize our output var
 43
        $html = '';
        # instantiate a new FileMaker object
        $fm = new FileMaker(FM_FILE, FM_HOST, FM_USER, FM_PASS);
 47
 48
        # get the record by it's id
 49
        $record = $fm->getRecordById($layout_name, $recid);
        # get the layout as an object
        $layout_object = $fm->getLayout($layout_name);
         # get the fields from the layout as an array of objects
        $field_objects = $layout_object->getFields();
 56
        # start compiling our form inputs
Shtml .= '<form action="'.$_SERVER['PHP_SELF'].'" method="post">';
Shtml .= "<input type=\"hidden\" name=\"process_form\" value=\</pre>
 58
"true\"
        />\n":
        $html .= "<input type=\"hidden\" name=\"recid\" value=\"{$recid}\"</pre>
60
/>\n":
        $html .= "\n";
61
         foreach($field_objects as $field_object) {
 63
             $field_name = $field_object->getName();
             # replace any spaces with underscores so field names match keys
in $_REQUEST arra
             $field_name_underscore = str_replace(' ', '_', $field_name);
 68
             # grab the field value from either the $_REQUEST array, or from
             if (array_key_exists($field_name_underscore, $_REQUEST)) {
                 if (is_array($_REQUEST[$field_name_underscore])) {
                      $field_value = implode("\n", $_REQUEST[$field_name_
underscore]);
                 } else {
 74
75
                         orah whatever was sent
                      $field_value = $_REQUEST[$field_name_underscore];
 76
77
 78
            } else {
                 # this must be the first time through the form because $_
REQUEST array does not exist for this field
```

LISTING 1: Continued...

```
$field_value = $record->getField($field_name);
83\, \, # get the style type, which will tell us if there is a value list attached to the field, and if so, what style
                       $field_style_type = $field_object->getStyleType();
  84
  85
                       # output the form control appropriate to the field style type
  86
  87
                       switch ($field_style_type) {
  88
                               case 'POPUPLIST':
  90
                                       # start compiling html for this select control
                                       $html .= "\n"
                                       $html .= "{$field_name}\n";
  92
                                        $html .= "\n";
  93
  94
                                       $html .= "<select name=\"{$field_name_underscore}\">\n";
  95
  96
                                       # loop through the values from the list attached to this
                                       $values = $field_object->getValueList();
foreach($values as $value) {
  98
                                               $selected = ($field value == $value) ? '
selected="selected"' : '';
                                              $html .= "<option{$selected}>{$value}</option>\n";
                                       # close the open tags
$html .= "</select>\n";
                                       $html .= "\n";
                                       break;
108
                               case 'CHECKBOX':
                                       # start compiling html for this checkbox set
                                       $html .= "\n"
                                       $html .= "{$field_name}\n";
                                       $html .= "\n";
114
                                       # loop through the values from the list attached to this
                                       $values = $field_object->getValueList();
116
                                       foreach($values as $value)
                                              $checked = (strpos($field_value, $value) !== FALSE)
? ' checked="checked"' : ''
119
                                               $html .= "<input type=\"checkbox\" name=\"{$field_</pre>
name\_underscore \cite{Constraint} [] \ value = \ \cite{Constraint} \ schecked \ /> \ value \ \cite{Constraint} \ schecked \ /> \ \cite{Constraint} \ schecked \ schecked \ /> \ \cite{Constraint} \ schecked \ schecked \ /> \ \cite{Constraint} \ schecked \ schecked
                                       # close the open tags
                                       $html .= "</select>\n";
$html .= "\n";
                                       break:
126
                               case 'RADIOBUTTONS':
                                       # start compiling html for this checkbox set
                                       $html .= "\n";
$html .= "{\field_name}\n";
                                       $html .= "\n";
                                       # loop through the values from the list attached to this
                                       $values = $field_object->getValueList();
136
                                       foreach($values as $value) {
                                              $checked = (strpos($field_value, $value) !== FALSE)
? ' checked="checked"' : '';
                                              $html .= "<input type=\"radio\" name=\"{$field_name_</pre>
138
# close the open tags
                                       $html .= "</select>\n";
                                       $html .= "\n";
                                       break:
                              default:
# the remaining field style types (EDITTEXT and CALENDAR) are best represented as text inputs
td>'."\n";
                                       break:
               }
                # add a submit button and close the open tags
```

LISTING 2

```
2 /* update_status.php */
  4 #######################
  6 #######################
 8~\# For security reasons, these lines should either be included from a 9~\# config file above the document root, or possibly captured during a 10~\# login and stored in the SESSION superglobal array
10 + Togrin and stored in the Session superg
11 define('FM_HOST', '127.0.0.1');
12 define('FM_FILE', 'ProductCatalog.fp7');
13 define('FM_USER', 'esmith');
14 define('FM_PASS', 'f!r3crack3r');
 16 # include the FileMaker API for PHP
 17 require_once ('FileMaker.php');
 19 # handler for showing, validating, and processing the form
20 if (array_key_exists('process_form', $_POST)) {
21    if ($errors = validate_form()) {
              $page content = show form($errors):
         } else {
 24
             $page_content = process_form();
 26 } else {
         $page_content = show_form();
 28 }
 FORM DISPLAY
 31 #
 32 ########################
 34 function show_form(\( \)errors = array()) {
         # initialize variables
         $layout_name = 'update_status';
         $post_manufacturer = (array_key_exists('manufacturer', $_POST)) ?
$_POST['manufacturer'] : '';
         $post_status = (array_key_exists('status', $_POST)) ? $_
POST['status'] : '';
         # instantiate a new FileMaker object
         $fm = new FileMaker(FM_FILE, FM_HOST, FM_USER, FM_PASS);
 43
         # create a new layout object
         $layout_object = $fm->getLayout($layout_name);
         if (FileMaker::isError($layout_object)) {
    return (''.$layout_object->getMessage().' (error '.$layout_
 46
object->code.')');
 49
 50
         # get the manufacturer value list as an array
         $manufacturers = $layout_object->getValueList('Manufacturer');
if (FileMaker::isError($manufacturers)) {
   return (''.$manufacturers->getMessage().' (error
 .$manufacturers->code.')');
        }
 56
57
         # sort manufacturers
         sort ($manufacturers);
 59
         # create the html manufacturer options
 60
         $manufacturer_options = "<option>Select a manufacturer...</option>\
n";
         $manufacturer_options .= "<option>-</option>\n";
 62
         foreach($manufacturers as $manufacturer) {
$manufacturer_options .= "<option{$selected}>{$manufacturer}
option>\n";
65
 66
         # compile errors as html, if any
$error_list = '';
 67
         if (count($errors)) {
              $error_list .= ''."\n";
              foreach ($errors as $error) {
    $error_list .= "{$error}\n";
 73
74
75
              $error_list .= "";
         }
         # insert the errors and manufacturer options into a form
         $html = <<<HTML</pre>
 79 {$error_list}
 80 <form action="{$_SERVER['PHP_SELF']}" method="post">
```

LISTING 2: Continued...

```
<input type="hidden" name="process_form" value="true" />
        <select name="manufacturer">
 83 {$manufacturer_options}
        </select>
        cy><input type="text" name="status" value="{$post_status}" />
cy><input type="submit" name="submit" value="Continue" />
 85
 87 </form>
 89 HTML:
 90
        return $html;
 91 }
 94 #
        FORM VALIDATION
 97 function validate form() {
        $errors = array ();
if ($_POST['manufacturer'] == 'Select a manufacturer...') {
   $errors[] = 'Select a manufacturer';
        if ($_POST['manufacturer'] == '-') {
            $errors[] = 'Select a manufacturer';
        if ($_POST['status'] == '') {
            $errors[] = 'Status is required';
        if ($_POST['status'] != strip_tags($_POST['status'])) {
            $errors[] = 'HTML tags are not allowed in the Status field';
        return $errors;
112 }
115 #
118 function process_form() {
119  # instantiate a new FileMaker object
        $fm = new FileMaker(FM_FILE, FM_HOST, FM_USER, FM_PASS);
        # set a couple variables
        $layout_name = 'update_status';
$script_name = 'Update Status';
124
        $script_param = $_POST['manufacturer']."\n".$_POST['status'];
        # call the script with the parameter
        $script_object = $fm->newPerformScriptCommand($layout_name, $script_
name. $script_param);
        # run the script
        $script_result = $script_object->execute();
        # check for errors
        if (FileMaker::isError($script_result)) {
  return (''.$script_result->getMessage().' (error '.$script_
134
result->code.')');
136
        }
POST['status']} status.
140 <a href="{$_SERVER['PHP_SELF']}">Click here to continue...</a>
142 HTML;
        return $html;
144 }
145
146 ########################
147 #
        HTML RENDERING
148 #########################
149 ?>
150 <html>
             <meta http-equiv="Content-type" content="text/html; charset=utf-</pre>
8">
             <title>update status</title>
            <style type="text/css" media="screen">
                body {font: 75% "Lucida Grande", "Trebuchet MS", Verdana,
sans-serif; text-align:center;}
                 a, a:visited {color: blue;text-decoration: none;font-weight:
bold:}
157 a:hover, a:active {color: blue;text-decoration: underline;font-weight: bold;}
                 input, select {width:260px;}
                 #container {width:400px;margin:0 auto;padding:20px;}
                 .errors {background-color:yellow;border:2px solid
```

Conclusion

I hope that this article has given you a taste for the rapid application development that is possible with FileMaker Pro, FileMaker Server Advanced, and the FileMaker API for PHP. No, FileMaker is never going to be an Oracle killer; but I can't tell you the number of times I have seen a "temporary" FileMaker solution bridge the gap for someone who was waiting for a SQL solution that ultimately never materialized. If you would like to look at the API code, currently at public beta status, you can download the FileMaker API for PHP at no cost from http://www.filemakertrial.com/php/default.aspx simply by filling a short form. If you would like to play around with this code, you will

need a copy of FileMaker Pro, and you will also need File-Maker Server Advanced. Neither are available for free, but you can get limited versions of each by joining the File-Maker Solutions Alliance (FSA). There is an annual fee for FSA membership, but the amount of free software offered to members would more than offset the membership fee. Please visit http://www.filemaker.com/developers/joinfsa.html for more information about joining the FSA.

LISTING 2: Continued...

JONATHAN STARK is the President of Jonathan Stark Consulting, an IT consulting firm located in Providence, RI. He consults a variety of clients from the creative industry including Staples, Turner Broadcasting, and Ambrosi. He has spoken at the FileMaker Developers Conference, is a Certified FileMaker Developer, and teaches training courses in both FileMaker and Web publishing. Jonathan is reluctant to admit that he began his programming career more than 20 years ago on a Tandy TRS-80. For more information, please visit http://jonathanstark.com.

LISTING 1: Continued...

```
$html .= ' <input type="submit" name="submit"</pre>
value="save changes" />'."\n";
        $html .= "\n";
$html .= "</form>\n";
         return $html;
159 }
161 ######################
162 #
          FORM PROCESSING
163 ##########################
165 function process_form() {
166  # instantiate a new FileMaker object
         $fm = new FileMaker(FM_FILE, FM_HOST, FM_USER, FM_PASS);
        # set a couple variables
$layout_name = 'edit_product';
         $recid = $_REQUEST['recid'];
         # get the layout as an object
174
         $layout_object = $fm->getLayout($layout_name);
176
         # get the fields from the layout as an array of objects
         $field_objects = $layout_object->getFields();
         \ensuremath{\text{\#}}\xspace loop through fields, pulling values from the \ensuremath{\text{\$\_REQUEST}}\xspace array
         $values = array();
         foreach($field_objects as $field_object) {
             $field_name = $field_object--getName();
$field_name_underscore = str_replace(' ', '_', $field_name);
             if (is_array($_REQUEST[$field_name_underscore])) {
                  $values[$field_name] = implode("\n", $_REQUEST[$field_name_
underscore]);
187
                  $values[$field_name] = $_REQUEST[$field_name_underscore];
        }
         # create a new edit command
```

LISTING 1: Continued...

```
$edit_command = $fm->newEditCommand($layout_name, $recid, $values);
194
        # execute the edit command
        $edit command->execute():
196
        $html = 'Record has been updated!';
        return $html;
199 }
201 ########################
202 #
        HTML RENDERING
203 #################################
205 ?>
206 <html>
        <head>
            <meta http-equiv="Content-type" content="text/html: charset=utf-</pre>
8">
            <title>edit_product</title>
            <style type="text/css" media="screen">
                body {font: 75% "Lucida Grande", "Trebuchet MS", Verdana,
sans-serif:}
                table {width: 600px; border-collapse:collapse; border-color:
#cccccc; margin-bottom: 10px;}
                th {padding: 3px; background-color: #DDD; text-align: left;}
                td {padding: 3px;}
                table, th, td { border:1px solid #ccccc; }
                a, a:visited {color: blue;text-decoration: none;font-weight:
bold;}
                a:hover, a:active {color: blue;text-decoration:
underline; font-weight: bold;}
            </stvle>
        </head>
        <body>
            <a href="view_products.php">view products</a>
222 <?php echo $page_content; ?>
        </body>
224 </html>
```

Deploying PHP Applications

by Jeff Moore

Ease of deployment is one of the biggest advantages of Web applications over desktop applications, but that doesn't mean it's foolproof. This month, Jeff Moore takes a tour around some of the pitfalls of application deployment and casts an eye over the PEAR installer's response to them.

TO DISCUSS THIS ARTICLE VISIT:

http://forum.phparch.com/366

ompared to desktop applications, the user interfaces for most Web applications are restrictive, slow, clumsy and limited. The palate of tools for building desktop user interfaces is expressive and deterministic. On the other hand, Web applications must paint from a limited palette of form controls and attempting anything advanced or unusual invokes the browser compatibility bugbear. Why, then, are Web applications so successful?

I have a one-word answer: deployment. Web applications are vastly simpler to deploy to a user base than desktop clients ever were. How much simpler? Look at the UI advances that we have been willing to give up over the last ten years in order to gain that simplicity in deployment. Yes, rich clients, meaning AJAX or Flash, are the rage these days. But the UIs that these solutions provide still don't match what was available on the desktop in 1995.

When I was working on ERP systems in the '90s, deployment was perhaps one of the most difficult problems we faced. It would be heartbreaking to talk to a user with a valid suggestion for improving the system. "Yes," I would say, "that is a great idea, but I can't do that." "Why not," they would ask, "it seems so simple to do." "Oh, it is", I would agree, "and it would only take me a few minutes to implement it, but the best I can do is to write it down and submit it to the requirements committee." At that point, the users were crestfallen. They knew that their idea, no matter how great it might be, it was unlikely to emerge from the formal prioritization process.

Ah, but I'm a sympathetic guy with access to the source code control system. I could have informally granted any given request. Unfortunately, my work was only a small part of the process of a change reaching that user's desk. After I made the change in source code control, it would

become part of the software release process. A documenter would stop by my desk for an explanation of the new feature. The testing team would have to test my change. The training team might also become involved. There could be interactions with other programs or facilities that didn't even use or want that feature. The change might require a special module for the system installer and upgrade application. When the facility rollout cycle began, trainers would be dispatched, and the feature would be installed at best many weeks after the request.

At worst, something would go wrong at any one of the many stages, and the question would come up about who checked in that unauthorized change. Or worse, a small change would interfere with the rollout of a much more important change. It happened. We used to figure that the deployment costs of a change were an order of magnitude more than the programming costs of a change. The risk of something going wrong during the deployment process was a significant damper on new changes. Self-evidently, this does not serve the users of the system.

We obviously want to maintain the changeability of our software systems. In this column I talk a lot about writing maintainable code as a way to be responsive to user change requests. But having a good deployment process is key to being able to rapidly change a system. It is not enough to be able to make a change fast. We also want to get that change in front of our users fast.

So, what is different about Web applications? The big differential is the client deployment cost. The desktop machines already come with the browser installed, and the upgrade cycle for that browser is largely independent of a specific Web application. Rolling out an application to a thousand desktops is a much more involved task than rolling one out to a server that services a thousand users. For many applications, that deployment simplicity is worth the cost of some user interface sophistication. For many applications, that deployment simplicity enables us to get new features in front of our users faster.

On the server side, there is little difference. Deploying to a Web server is hardly different than deploying to an application server. I've found that many of the same deployment processes and techniques apply.

Save the Code, Save the Project

The foundation of any deployment process is your source code control system. Source code control makes the difference between chaos and order. With source code control, your deployment process can be controlled, deterministic and safe. Without it, you may never know exactly what is deployed. Or worse, you may lose code, be afraid to make changes, or have to undergo a time-consuming comparison process at deployment time. Losing source code is not very professional. The medical community has a word for this. They call it malpractice. There is a legal term, res ipsa loquitur, which means the thing (or result) speaks for itself. If a pair of forceps is found inside a patient after surgery, malpractice is assumed. In the same way, if you misplace source code, malpractice can also be assumed: res ipsa loquitur.

I favor subversion these days. It is the easiest SCM to use that I have tried, in 15 years of using SCM. It is free, and it is open source. It is easy enough to install and with the availability of hosted source code control solutions, there is no excuse for not using source code control.

Work with a Net

Early in my career I was working in an office with a server that had mirrored drives. One drive failed, so a network engineer came to work on it. To fix it, he was going to reformat the bad drive and re-mirror the drives. Just before he hit enter on the format command, I asked him, "Are you sure you have the right drive?" "Yes, I'm sure," he confirmed. I think you can guess what happened next. Another time, during an ERP rollout, a bug in an inventory allocation algorithm made it into production. Everything was fine, except that when a semi truck left the facility, the system would exchange the record of the inventory that went out on the truck with a random inventory item from the warehouse. Even though the right products were placed on the truck, we had no accurate record of what was shipped and the picking algorithms from the warehouse weren't right. By the time the problem was diagnosed and repaired, a thousand tons of products had been misplaced. The forensics, data repair and inventory auditing took weeks afterward.

I have a million stories of the fallibility of computerized systems and the humans who build and operate them. The point is that things go wrong. Worse, they often go wrong at deployment time. The first lesson is that you never want to deploy anything into a 24*7*365 environment on a Friday, unless you plan on working over the weekend. The second lesson is that you always want to have a backup, and a rollback contingency plan.

Automate This!

The key to fast and safe deployment is automation (and testing). As with my hard drive formatting friend, people

make mistakes. By automating the deployment process, we reduce the chance of making those mistakes. An automated process can be tested. A dry run can be done. If something does go wrong, we can examine the automation script to see what happened. Once the process is set up, automated deployment reduces the labor costs of deploying an application. When deploying an application is easy and safe, there is incentive to do it more often. This is a good thing. There are several different ways to automate the process of installing files onto a production server. One way is to build a set of update scripts around the rsync utility or some other file synchronization utility. This is fairly easy to set up, and I have to admit that this is the primary method that I use.

However, there is what I would consider a better way. The PEAR installer, designed for exactly this purpose, has become better and better at installing applications and libraries, as well as increasingly open. The PEAR installer may be the de facto standard for the installation of PHP applications. Greg Beaver, the primary PEAR installer developer, has literally written the book on the PEAR installer, called "The PEAR Installer Manifesto". I heartily recommend it.

What are the advantages of the PEAR installer? PEAR provides a standard packaging mechanism, along with tools to create those packages. It also provides a mechanism for distributing these packages via its channel mechanism. Channels can be private and authenticated. The PEAR installer has good support for describing and managing the dependencies between packages. The PEAR installer can run post installation scripts. This is extremely useful for data format migrations.

It's fairly easy to set up scripts to transfer files, either with rsync or with the PEAR installer. Data format migrations are more difficult. If we make a change in the database structure, we want to automate the process of migrating that change onto our production server. We don't want to do the equivalent of formatting the wrong drive when typing SQL into the database by hand. Even



www.dynamicwebpages.de sex could not be better | dynamic web pages - german php.node

news . scripts . tutorials . downloads . books . installation hints

with backups, large systems can take a long time to restore and downtime can be expensive.

The database update portion of our old ERP installer dramatically reduced deployment errors and dramatically increased our deployment confidence when it was rolled out. We checked our database schema into our source code control system. The installer script generator program would examine the table structures in source code control. It 'understood' the tables and fields, and was able to detect the difference between different revisions and generate an upgrade script to install that change (and a downgrade script as well). It was also possible to check in standardized rows for some tables. We would review the generated script, and it would go through a testing process.

The same type of system is possible with PEAR's post-installation scripts. I have to admit that I am not yet using the PEAR installer, although I think that it is the way to go in the future. There is still a bit of a learning curve (lower now, with the release of Greg's book). It still takes a bit of effort to set up the scripts to create the PEAR packages. I also don't know of any PHP based SQL diff tools similar to those we used on our ERP project. However, I expect all these obstacles to fall as more and more developers and projects switch to using the PEAR installer.

Every File has a Role to Play

There are several considerations when installing files onto a Web server. Different files have to go into different locations. The major division between locations depends on whether the file should be public and go in the Web server's document root, or whether a file should be kept in a location that is not publicly accessible. Files can go in the front end or back end. Generally, it is best to minimize the number of files that are installed into the front end. For security purposes, we don't want to make any files publicly accessible that aren't required to be so. I generally prefer to put only a small bootstrap front controller file into the document root, and funnel all the dynamic content requests for the application through that file.

Another consideration in determining the location of our application's files is the modifiability of a given file. We may wish to segregate files that aren't normally modified by the application. Changes to these files would usually be made into source code control first, and then deployed to the server. An application may be installed on more than one server, and these files would not be different between servers. The PEAR installer has a special file role for these kinds of files, called the php role. For static data, PEAR's data file role may also be appropriate.

Some files are changeable, and may be generated by the application. A good example of this would be cached files, which can usually be regenerated if they are deleted. It can be a good idea to segregate these files into their own directory. Deleting all the files in that directory could clear the cache.

Another issue is configuration data; that is, data that might be specific to a particular server or a particular installation of the application. We don't want to hard code this data into the static application files. This makes it more difficult to manage a development or testing server, or multiple installations of the same application. It may be a good idea to segregate these kinds of files. I try to minimize configuration data. I tend to try to put it into the bootstrap loader that goes in the public document root. The PEAR installer has some specialized support for this kind of data. It can replace configuration data in specific files.

Perhaps some data will be modified on a per-installation basis, but isn't in your source code control for one reason or another—for instance, data that crosses organizational boundaries. A good example of this might be when a specific directory is created to hold templates.

There is another file location consideration for publicly accessible files. Sometimes we might want to segregate our static public files from our dynamic public files for performance reasons. Our deployment process should be able to handle this.

Yin and Yang

On the server, we segregate and organize our deployed files by type and by modifiability. This begs the question, how do we organize our files during development? One choice is to mirror the organization of the deployed application. In this style, the first level of organization sorts the files into type. You often see this in various frameworks. There is a top level tests, view, template, model, or controller directory. This is a good scheme if you are using the file synchronization method of deployment, and it is guite common. The drawback to this style of organization is that we often work with related files of the same type. We may just have a PHP file and a test file, or a PHP file and a template file, or something more complicated. When different files from our working set of files are in divergent directories, working with those files can become cumbersome.

This suggests an alternative approach to file organization. We can organize at our top level by package, perhaps even by PEAR package. We can still organize by type at a lower level, perhaps using file extensions to distinguish types. Having all our working set of files in the same directory makes it much easier to work with these files.

The drawback of organizing by package is that it introduces two different file organization systems. Our development environment and source code control system group by package, while our deployment environment uses the groupings that make most sense there. If you are using the file synchronization method of deployment, this may be too much to handle. However, the PEAR installer method of deployment can easily translate between the two different methods of file organization.

The one thing we absolutely do not want to do is to insert a deployment phase into the development cycle. Part of the power of PHP is that it gives immediate feedback during development. We make a change during development, and can immediately refresh the page in the browser to see the results. We don't want to interfere with that by requiring some kind of deployment or build script. If we have two different file organizations, then it becomes more difficult for the application to locate its files as it runs. Ideally, the application should be able to locate its files in either configuration—but having the ability to do so complicates the application itself.

I certainly don't have all the answers for deployment. I've been moving toward a package based, rather than a type based organization. I like having related files co-located. I've also been moving toward using the PEAR installer, although I don't yet have all the tools in place for this. I have yet to resolve the issues that arise when locating supporting files in dual configurations.

JEFF MOORE learned to program in the 80's, worked on ERP systems in the 90's and is devoting this decade to PHP. Jeff does freelance programming, works on the open source framework WACT and occasionally posts to his blog at http://www.procata.com/blog/.

Want YOUR 15 minutes of



php | architect wants to hear from you!

If you want to bring a PHP-related topic — be it your personal research, company software, or anything else the professional PHP community might be interested in — 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!

Why upgrade to PHP 5.2.1?

by Ilia Alshanetsky

A few weeks ago, I rolled PHP 5.2.1—the latest word in PHP development. How is this relevant to a security article? Well, the 5.2.1 release is unusual in its focus on security—and, in particular, in the number of security issues it addresses. This is an overview of the security fixes and enhancements found in PHP 5.2.1. I hope that it will convince you of the need to upgrade your PHP deployment.

hen it comes to security faults in PHP, the first thing we need to do is classify the kinds of exploit so that we can distinguish between them. We do this by looking at the way in which an exploit can be abused.

In some instances, the exploit can be trigged remotely. This is considered exceptionally dangerous, since it would allow an external hacker to trigger undesired behavior on any PHP enabled server. Fortunately, this particular type of *remote exploit* is very rare in PHP. In the past, this kind of security hole has generally been caused by poorly coded input processing mechanisms, such as file upload handlers.

Another type of exploit is *function specific*. This type of exploit can be triggered externally, but only if the user's code relies on a particular function or extension containing a bug. Often it is *only* possible to trigger this kind of exploit remotely when the user code exposes external input directly to the vulnerable function. One such instance would be the use of the unserialize() function on user-supplied input without any validation.

TO DISCUSS THIS ARTICLE VISIT:

http://forum.phparch.com/367

unserialize(), being designed specifically for internal use, lacks some of the protection available in other functions. Therefore, it could be easily abused to do all manner of nastiness.

Finally, there is the *local exploit*, which requires the attacker to have local access to the server in order to be exploited. Generally, this kind poses the biggest headache for ISPs that need to deal with hundreds of local users, some of whom are not looking to play nice. It should be noted that these local exploits could become remotely triggerable, given a PHP application that allows code injection. This type of exploit is unfortunately quite common; a good example would be a safe_mode or open_basedir bypass.

Now that we've got the exploits classified into types, let's move on to examine the actual issues that were fixed in PHP 5.2.1.

Remote Exploits

The good news is that there were no purely remote exploits discovered, meaning that a generic install of PHP is not vulnerable out of the box. However, a series of issues were found and resolved that could result in partial remote exploits, stemming from bugs inside commonly used functions that are all too often exposed to unfiltered user input.

As I mentioned earlier, one such exploit could be found inside the unserialize() function when deployed on 64-bit systems. A user-supplied string could potentially trigger an un-terminated loop that will occupy the process until maximum execution time is reached—about 30 seconds on most systems. This meant that, with as few as ten requests, an attacker could launch a fairly successful denial of service attack against a 64-bit server. It should be stressed that the unserialize() function was never intended to deal with user supplied data; its goal was to decode internal serialized format data stores inside files, databases or shared memory. As such, any use of the function with user supplied data is a dangerous abuse of the feature, and will result in a local exploit becoming a remote one.

If an application absolutely *must* pass serialized data via GPC mechanisms, a simple HMAC hashing algorithm should be used to validate the transferred data and ensure that it was not tampered with along the way. The example below demonstrates this kind of protection in action:

```
/* output block */
echo '<input type="hidden" name="s" value="'.($out =
serialize($data)).'" />';
echo '<input type="hidden" name="hash" value="'.
hash_hmac('md5', $out,
   'secretkey').'" />';
/* input block */
if ($_POST['hash'] == hash_hmac('md5', $_POST['s'],
   'secretkey')) {
        $data = unserialize($_POST['s']);
} else {
        exit("Hacking attempt!");
}
```

In this example, before unserialize() is called on the data it first must pass through HMAC validation, which uses the md5 algorithm to ensure the data was not modified by the user. The secret key ensures that a hacker can't simply calculate an md5() hash of newly corrupted data and bypass the validation.

The next semi-remote security exploits were found inside the filter extension, which is a bit ironic given that the extension's job is to protect scripts against invalid user input. A bug was discovered inside a combination of string filters designed to strip high-ASCII characters and characters that could trigger XSS, such as <>. The problem was caused by the fact that the filter was not continuous, thus allowing the hacker to inject

invalid input by creating certain character sequences, which would actually *create* XSS on stripping. This issue is somewhat similar to the <code>mysql_escape_string()</code> bug that I explored a few months ago in this column. Another filter validation issue was found inside the IPV4 validation code, which was not binary safe. This would allow a hacker to trick the validation with just a few carefully placed <code>\0 (NULL)</code> bytes. Finally, there was an underflow inside the <code>trim()</code> implementation within the filter extension, which arguably is the most dangerous of the exploits listed here. On PowerPC systems such as G4 Macs, this could be used to trigger arbitrary code execution; on other systems, it 'only' crashes PHP.

Aside from upgrading your copy of PHP, there is no alternative solution to the filter problems. Fortunately filter is a very new extension, having only appeared in PHP 5.2.0, so the number of affected users is relatively low.

Local Exploits

On the local exploit front the situation is much worse then with remote exploits, simply due to the sheer number of issues that were found and resolved. The most common problem in PHP's internal code base appears to be a buffer overflow. A buffer overflow comes about when a memory buffer is allocated that is too small to store the data it is intended to contain. It leads to potential arbitrary code execution, and at best either a crash or memory corruption. Buffer overflows were found in several extensions and functions, with the worst culprits being the IMAP and Interbase extensions; between them, those two have amounted for over half a dozen such holes. The stream filter functionality was not immune from the problem either, with stream_filter_append() and the filter factory code containing one bug apiece. Finally, the str_replace() function could experience the same problem when faced with very large strings, as did the binary string decoding inside the SQLite and pdo_sqlite2 extensions.

As far as a solution goes, if you use those particular extensions, or are running an ISP, your mitigation strategy should be to upgrade your PHP installation as soon as possible. That said, it would be difficult to avoid using str_replace(), and nor would anyone in their right mind disable it, so the same advice goes for all.

The next issue is also a buffer overflow, but this one demands a separate note due to its nature. In PHP 5.2.0, a new memory manager was introduced into PHP. It brought with it many benefits and speed improvements; however, it also brought one subtle change that has security implications. In the past, if a negative size was passed to PHP's memory allocation macros, PHP would error out with a message saying that it couldn't allocate

a block of memory of that size. Since the new memory manager was introduced, instead of returning an error, PHP 5.2.0 instead returns a minimum block size. In other words, if, somehow, a negative value such as -100 was passed to PHP's internal allocation macro emalloc(), it wouldn't fail but would rather allocate a very small block of memory. When code attempts to store some data into memory—something that often exceeds the minimum block size—a buffer overflow would be triggered. The

code block. These include two distinct ways to bypass safe_mode and open_basedir restrictions via the userland code responsible for setting the session storage directory. This is a relatively minor exploit, as it only provides the hacker with limited read/write capability. It could only be used by a local user to access a session belonging to another local user, on a system running PHP as an Apache module. A non-upgrade solution to this problem would be to disable the session_save_path()

The good news is that there were no purely remote exploits discovered.

most common cause of this particular buffer overflow is when allocation is based on a product of two numbers on 32-bit systems, where the product is greater then 2.14 billion. The resulting integer overflow mutates a very large number into a very small number—for example, 4294967196 would become -100.

An extensive code audit was performed, and many instances where this could occur were identified. The vulnerable areas were changed to use safe_emalloc(), a special allocation macro that was specifically designed to defend against this problem. However, affected code was found across several PHP extensions, so once against the safest course of action is to upgrade.

The next set of issues was caused by something known as heap exposure. This is a security vulnerability whereby a user can inject data into a known heap block, allowing for arbitrary code execution. It is triggered primarily through the session extension, and mainly involves making \$_SESSION contain _SESSION or by-reference references to \$_SESSION. Overwriting this causes heap corruption/exposure, which in turn makes it possible to execute arbitrary code. To exploit this issue, the hacker would generally need local access to the machine, so it falls into the local exploit category.

Aside from upgrading, the only way to defend yourself against this security hole would be to disable the session extension until you are able upgrade your PHP installation.

While we're on the topic of the session extension, there are three more issues affecting that particular and ini_set() functions, thereby preventing the user from modifying these values. An ISP could still offer the option of a dedicated session directory for each user, by setting the session.save_path value on a per-virtual host basis via the Apache configuration file.

The final issue in the session extension can be traced to the binary serializer, one of the means by which PHP variables can be encoded into a transportable format that can be stored inside a database, filesystem and memory. The binary serializer is a little more economical in terms of size then the default string-based serializer, but despite that advantage is rarely seen in a production environment. In fact very few people even know it exists, which means that very few people are affected by this particular bug. The problem in this code was due to its handling of high-ASCII values (127 and above). Improper handling of these values could lead to a potential exploit, allowing the attacker to execute arbitrary code by writing to random memory addresses—a less then desirable mode operation by most accounts. Until you upgrade to PHP 5.2.1, the simplest solution for the binary serializer problem would be to switch to using the php serializer.

Moving on to the shmop extension, designed to provide a simple interface to shared memory, an issue was discovered whereby the extension failed to validate the type of resource used in its write and close operations. This meant that, by passing a non-shmop resource to the extension, it was possible to write and read directly to/from the memory block pointed to by the resource. Gain-

ing the ability to write arbitrary data to memory makes it possible for the hacker to perform arbitrary code execution, causing all manner of problems. Fortunately, for this to happen, our hacker would need to have permission to run code on the server, since the resource parameter for the shmop functions does not come from user input. Possible abuses of this vulnerability can include safe_mode/open_basedir bypasses, and also the execution of otherwise disabled functions. Since PHP does not have any mechanism for checking whether a resource is of a particular type from within PHP code, the only solution for users of ext/shmop is to upgrade to either PHP 5.2.1 or PHP 4.4.6.

Back in the PHP core, an additional security problem was found in the header() function, in particular when the function is executed with a header containing nothing but spaces. When this happens, the internal code triggers a fairly rare type of an exploit, called a *buffer underflow*. On most machines this will result in a simple crashes, but for users of PPC and Sparc it can lead to exploitable code execution.

As with the shmop exploit, the header() issue requires that the attacker has full control over the input value of the header() function, which usually would limit the exploit to local users. Using Google's code search facility, we were only able to locate two or three instances where this is done out in the wild. In the code we found, the value of the header was supplied entirely through user input, rather than being concatenated to another value. Taken alone, the header() exploit is fairly limited in value; and to be truly dangerous, it requires a specific brand of computer. All in all I would not be too concerned about the issue, but it does highlight the need to be careful in your usage of the header() function. A user-supplied value there will result in arbitrary header injection, even under the best of circumstances.

An additional subset of security issues that was found and resolved in the PHP 5.2.1 release includes the so-called *string format vulnerabilities*. These are often found inside the various *printf() functions, and stem from invalid parameter usage. The abuse of those exploits could permit a hostile user to perform arbitrary command execution with the privileges of the user when the PHP interpreter is executed.

In most cases these exploits can only be triggered locally, as they require the execution of PHP functions such as odbc_result_all() along with a certain set of user supplied parameters designed to trigger the function's weakness. In this particular case, the attacker would try to have the cumulative length of user-supplied parameters exceed the internal storage buffer assigned to store their values. When that happens, arbitrary code execution is made possible.

Another attack vector aimed at exploiting this set of

vulnerabilities is specific to 64-bit systems. There, by supplying an invalid string definition as the first parameter passed to the printf() function, an integer overflow can be triggered. Because the internal code lacked validation for values less then zero, it did strange things in these situations. Given carefully crafted code, those 'strange things' could enable an attacker to trigger code execution.

Normally this would be considered a local vulnerability, since you would not expect a developer to put user input into the string definition block. Unfortunately, a quick Google Code Search reveals that this actually happens relatively often, which means people running such code on 64-bit machines are potentially vulnerable to remotely triggered exploit. It is imperative that those using 64-bit machines either upgrade to PHP 5.2.1 or audit their *printf() PHP code to ensure that there are no instances where user supplied values are part of the string definition.

One of the new additions in PHP 5.2.0 was the new zip extension, which provides a means to read and create zip files. One of the convenience features introduced by the new extension was the stream filter zip://, which simplifies operations with zip files.

Unfortunately the code was a bit raw, and when it came to validating the zip:// stream filter the extension did not have proper handling for long file paths. This meant that code that uses external input to populate any of PHP's file opening functions could be vulnerable to a buffer overflow, which could then lead to arbitrary code execution, given a hostile local user.

That said, the zip extension was not enabled by default, and so the number of people for whom this presents a problem is quite limited. To check whether your PHP setup is vulnerable or not, you can run the following script:

```
if (function_exists('stream_get_wrappers') &&
in_array('zip',
stream_get_wrappers()) && version_compare(PHP_VER-
SION, '5.2.1', '<')) {
    echo "You need to upgrade!";
} else {
    echo "All's well.";
}</pre>
```

If you cannot upgrade, another alternative would be to recompile your PHP 5.2.0 installation without zip support.

New security features

Security fixes aside, the PHP 5.2.1 release also introduces new security features designed to improve the security of PHP itself and the applications running on top of it. The first of these improvements is an addition of the do-not-index header to the phpinfo() output, which

should prevent obedient search engines from indexing the page. This should prevent massive information disclosures caused by people leaving the phpinfo() page on their live sites, which currently results in those pages being indexed by search engines. This information leak allows the hacker to gather a plethora of information about the execution environment and PHP itself, making it much easier to attack PHP applications running on that server. Additionally, phpinfo() output includes all the INI and environment settings, which means that any passwords stored in INI or server environment settings are exposed for all to see on such a page.

A quick search on Google reveals that this is a major problem, as there are quite literally tens of thousands of indexed phpinfo() pages, presenting a treasure trove of information for statisticians and hackers alike. If upgrading is not an option, you may want to make sure that your site does not have any phpinfo() pages; or, better yet, disable the function entirely via the disable functions INI directive. The latter is a particularly useful option for hosting providers, who are not in control of the scripts executed by their users.

Another new feature is built-in heap protection, which previously was available only through add-ons such as the Suhosin extension. This mechanism provides a fail-safe for things like heap corruption in existing code, making vulnerabilities in PHP harder to exploit with the goal of executing arbitrary code. In a way, this feature is a safety net that will protect you against future instances of a function or functionality compromising a PHP installation.

The final security enhancement was that memory_ limit, until now an optional feature in PHP, is always enabled as of PHP 5.2.1—albeit with a very generous maximum limit of 128 megabytes. This change will quard against runaway scripts that might, intentionally or otherwise, try to perform a denial of service by taking up all the available memory on the system. It can also work as a buffer against attacks that rely on huge blocks of data, or exceptionally complex data structures such as deeply nested arrays. For better protection, it is recommended that most people should lower the default value to something more reasonable, in the region of 16-32 megabytes. PHP's own default is only so high due to the necessity of avoiding breaking existing applications.

The above summary is a quick overview of the key issues resolved by the PHP 5.2.1 release, and hopefully has given you a better idea of where you stand, securitywise, with your current PHP installation. Overall, though, it is the strong recommendation of the entire PHP development team that people upgrade their installation to PHP 5.2.1 or to PHP 4.4.6, which includes most of the protection mechanisms offered by the PHP 5.2.1 release.

ILIA ALSHANETSKY is the principal of Advanced Internet Designs Inc., a company specializing in security auditing, performance analysis and application development. He is an active member of the PHP development team, with hundreds of bug fixes to his name as well as a sizeable number of performance tweaks and features. Ilia is a regular speaker at PHP-related conferences worldwide, and has authored the php|architect Guide to PHP Security as well as several articles. He maintains an active blog at http://ilia.ws, which is filled with tips and tricks on how to get the most out of PHP.

ST, ND, RD,

omniti.com/careers M, Σ , 8,

1:F, 2:1E, 3:2D,

Scalability. Performance. Security.

PHP TRAINING SHOULD NOT BE COMPLICATED OR EXPENSIVE

Live Interactive PHP TRAINING PHP 5 Essentials

PHP ESSENTIALS

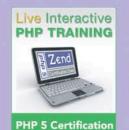
Dive into PHP with this great introductory/mid-level course designed for those who are approaching PHP for the first time or are moving to it from another language

Just ask the 600 students we trained last year.

Training is essential to stay on top of your game—but it can be expensive, cumbersome and inconvenient. Take learning to a new level with our exclusive live online training course and get the best training at the most affordable price!

- Better Than Live™: full classroom experience with the convenience of the Internet
- · Experience live training sessions with voice and screen sharing
- · Interact with your instructor and fellow students in real time
- Classes are spread across longer periods to ensure optimum learning
- · Includes free tutoring

Enjoy affordable prices without sacrificing the quality of your training—try us out today!



CERTIFICATION

This great mid-level course prepares you for the Zend Certification exam-and it comes loaded with over \$500 in free goodies!





Professional PHP 5

PROFESSIONAL

Designed for the seasoned developer who has mastered PHP, this course tackles the most advanced PHP 5 topics, like security, XML, design patterns, and much more.

php | architect

Free no-nonsense trial





///exit(0);///

Is Google Really the Best We Can Do?

by Marco Tabini

azine can be unaware of Google's enormous success—not only from a financial perspective, but also from the point of view of popularity and penetration of our culture. After all, I am quite sure that the postal service does not deliver magazines under rocks.

Google is clearly a very successful company; its revenues are quite impressive, as is its growth and, above all, the enthusiasm that it generates in users and investors alike. It is curious that I have seen no-one ask themselves that most interesting of questions: why?

A successful business is, in many ways, a reflection of the society it exists in. During World War II, successful businesses sprung up well outside the traditional military complex. The Higgins Corporation of New Orleans, for example, became very successful through producing landing craft used during the invasion of Europe. Once the war was over, the company ceased to be useful; it eventually went bankrupt.

It stands to reason that Google has become successful because it has filled—and continues to fill—a

need that our society expresses. And that, to my way of thinking, speaks very poorly of our society.

Google's express goal is "to organize the world's information." The company does not create anything, much less new content. In fact, it survives only because it has unlimited access to content created by others. If Google were forced to pay in order to use other people's content—you know, the way just about everyone else does... there would be no Google.

It's arguable that Google creates new technology. Some of the research the company has done—for example, on distributed computing and large networks—has been widely publicized and heralded as a breakthrough of ingenuity. Still, none of the applications that Google have created has (yet) had a significant impact on our society, with the honourable exception of their search engine. Consider, for example, Gmail. When it was introduced in "beta," almost everyone seemed to think that it would have been a very disruptive product—but for all the wrong reasons. Who cares if you get a full GB's (now several) worth of free e-mail space? The real innova-

tion in Gmail is in its user interface—not because it was one of the first fully-featured AJAX applications, but because it forces a distinct change in the way one deals with e-mail. Gmail's "labels," which replace folders in other MUAs, have hardly gone noticed. Thus, the disruption introduced by Gmail has been largely absorbed by the market: since everyone seemed to think that storage was the key, all the other vendors matched Google's capacity, or exceeded it. I have a hard time understanding why everyone seems to be so impressed by Google, the company. It is scary to think that we consider a company whose ultimate objective is nothing more than to catalogue knowledge as one of the biggest success stories of our times. The role they fulfill is undoubtedly important; human knowledge (including its spam and YouTube clips) has reached such huge proportions that we need a mechanism that allows us to retrieve information in a timely manner (whether Google allows you to do that or not is another matter).

However, Google's corporate mission is, for lack of a better word, pathetic. Their aim is not to make our world better, but merely possible. Ask yourself: if Google were to fulfill its mission... how would we be better off?

This attitude is part of a horrible trend that our entire society has become slave to. The mission of our universities is no longer to teach, as can be demonstrated by the fact that our governments are making it less and less possible for those without money to attend. Universities are big on "research," but only when it is conducted for a specific, known purpose. However, as Isaac Asimov once pointed out, "the most exciting phrase to hear in science, the one that heralds new discoveries, is not Eureka! (I found it!) but rather, 'hmm... that's funny..." By asking certainty of a process that is by definition uncertain, we are limiting our potential and damaging our future.





NEXCESS.NET Internet Solutions 304 I/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



\$6⁹⁵/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 \$2|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 I \$ 695/mo

900 MB Storage 30 GB Transfer Unlimited MySQL Databases

Host 30 Domains PHP5 / MYSQL 4.1.X NODEWORX Reseller Access

21

NEXRESELL 2 \$5995/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





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

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





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

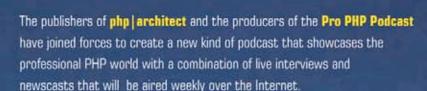
the audience is listening

php | architect's

PRO: PHP podcast







php | architect's Pro PHP Podcast airs weekly, alternating between a pre-recorded newscast and an interview, which are broadcast in real time using php | architect's exclusive webcasting technology.

Members of the audience are welcome to participate in the live interviews, where they have the opportunity to ask questions in real time. All podcasts will be subsequently made available for offline download either through a web browser or using most RSS-compatible podcasting applications, including Apple's iTunes.

be part of it!

php | architect

http://podcast.phparch.com