Easy PHP with the eZ Components library

TAKING IT EASY

eZ Components is a free enterprise-level PHP library that uses state-of-the-art language features. We show you the library's potential on the web and at the command line. **BY TOBIAS SCHLITT**

The rise of PHP has spawned a new industry in tools for PHP development. By providing standard, reusable components, these tools simplify development and speed up the coding process. One of the entries in this race is eZ Components, a generic PHP library created by eZ Systems, the makers of the eZ Publish Content Management System. The eZ Components library, which is available for download under the open- source New BSD license, offers an easy-to-use collection of standard components designed to reduce errors and save development time [1]. We demonstrate a few interesting features of the eZ Components library and put those features to work on a sample application that forwards email messages.

The individual components that make up eZ Components are independent for the most part, although they support collaborative use and integration via the tie-in model. (A tie-in is a component that links to otherwise independent components.) The eZ Components developers extensively document even the most trivial function in the code. For example, you will find a comprehensive tutorial for each component and sample applications to match.

PHP specializes in web applications, so the eZ Components library comes with a large selection of web tools. A template engine allows for easy creation of HTML templates and abstracts the display logic from the remaining program structure.

The syntax, which is reminiscent of the Smarty template system, is fairly simple to learn. An integrated caching mechanism that builds the templates into the PHP code on first use keeps performance hits to a minimum. Automatic escaping of all output provides protection against XSS attacks. Plus, the template component has a flexible interface for extensions that supports the integration of new functions and user-defined block structures.

The *UserInput* component, which is based on the filter extension included in PHP 5.2, supports verification and validation of user input. Fans of the model-view-controller approach will find the URL component a flexible tool for URL processing. The *Feed* package gives developers the ability to handle and create various XML feed formats. The component is currently in alpha.

In addition to these web-only components, eZ Components contains many packages that are useful for other applications: reading and writing configuration files (*Configuration* component), creating, unpacking, and manipulating archives (*Archive* component), logging in various formats (*EventLog* and *EventLogDatabaseTiein*), and more.

One of the library's major focuses is database abstraction. Users are free to choose which of the three abstraction layers they want to use. The *Database* component provides a wrapper for PHP's own database abstraction, PDO, which standardizes the fairly quirky exceptions thrown by PDO.

Besides this, *Database* has a second abstraction layer, an SQL generator that uses calls to methods to create SQL statements independently of the underlying database. This supports transparent porting of applications from, say, MySQL to Oracle.

The third abstraction layer is handled by the *PersistentObject* package, which implements object relational mapping (ORM), giving users the ability to store, load, and manipulate PHP objects directly in and from the database. In con-

Listing 1: Integrating eZ Components

```
ini_set( "include_path", "/
   dev/ez/ezcomponents:." );

require_once "Base/base.php";

function __autoload(
   $className )
{
    ezcBase::autoload(
   $className );
}
```

Listing 2: eZ Components at the Console

```
$output = new ezcConsoleOutput();

$output->formats->error->color = "red";
$output->formats->error->style = array( "bold" );
$output->formats->error->target = ezcConsoleOutput::TARGET_STDERR;

$input = new ezcConsoleInput();

$options["help"] = new ezcConsoleOption( "h", "help" );
$options["help"]->isHelpOption = true;

$options["server"] = new ezcConsoleOption( "s", "server" );
$options["server"]->type = ezcConsoleInput::TYPE_STRING;
$options["server"]->shorthelp = "The server to fetch mail from.";
$options["server"]->longhelp  = "The POP server to fetch mail
   from.";
$options["server"]->mandatory = true;

// ...

$options["verbose"] = new ezcConsoleOption( "v", "verbose" );
$options["verbose"]->type = ezcConsoleInput::TYPE_INT;
$options["verbose"]->default = 1;
$options["verbose"]->shorthelp = "The verbosity of this program.";
$options["verbose"]->longhelp = "Verbosity of 1 is standard and
   means   normal info output. 0 will switch info output off.";
```

trast to similar implementations of other libraries, *PersistentObject* does not require the PHP classes of a persistent object to inherit from a defined standard class. This makes it possible to configure arbitrary objects as persistent without affecting the inheritance hierarchy.

The *DatabaseSchema* component is also for database operations; it supports abstract storage of database structures and creation of structures in a database. All of the database components currently support MySQL, PostgreSQL, SQLite, and Oracle. The next version will have support for Microsoft SQL Server.

Graphics

eZ Components also has components for handling graphics. The *ImageAnalysis* and *ImageConversion* components analyze and convert image files, making it easy to scale and crop images, change color spaces, and convert between formats with just a couple of steps. Effects give users the ability to create thumbnails or add borders, noise, swirls, and watermarks. Both components are modular and can use the GD PHP extension, as well as ImageMagick back-ends.

The *Graph* component handles the visualization of static data, generating line, bar, and pie charts. There is no need to do without 3D look or highlighting effects (which you may be familiar with from various office packages). Any output format supported by the GD extension is fine, as are the SVG XML vector format and Adobe Flash (via the PHP Ming extension).

Finally, let's not forget the *ConsoleTools* component, which is the basis

Listing 3: Handling Program Options

```
foreach( $options as $option )
{
    $input->registerOption( $option );
}

try
{
    $input->process();
}
catch ( ezcBaseException $e )
{
        $output->outputLine( $e->getMessage(),
 "error" );
    die( -1 );
}

if ( $input->getOption( "h" )->value !== false )
{
    $output->outputLine( $input->getHelpText(
 "Forwarwds mail from the console." ) );
    exit( 0 );
}

$output->options->verbosityLevel = ( $verbosity =
$input->getOption( "v" )->value ) === false ? 1 :
$verbosity;
```

for the sample application. *ConsoleTools* provides classes that format output on the shell, along with a mechanism for handling command-line parameters.

Installing eZ Components

The easiest way to install eZ Components is to use the Pear Installer, included with recent versions of PHP:

```
pear channel-discover ➘
components.ez.no
pear install eZComponents
```

If you prefer not to run the Pear Installer, you can download the current eZ Components package from the website [1], and unpack it. Also, you will need to enable the eZ Components autoload mechanism, which automates access to all component classes. Listing 1 shows the code for this. The only class you need to add manually is the main class for the *Base* components. eZ Components handles the rest automatically. Don't forget to modify the *include_path* entry in Line 1 for your own system.

To demonstrate the eZ Components library, I'll create a simple program that forwards email. The program expects various command-line parameters, which it uses to establish a connection to an IMAP server, retrieve the messages from the server, compact the messages to a digest, and delete the messages. Add a cronjob, and this would give you a neat way of retrieving messages from a second mailbox and forwarding them to your main mailbox.

Listing 2 first initializes an *ezcConsoleOutput*-type object, which later handles the text output. The script defines a new format called *error* in the object's *formats* attribute to enable bold, red type for standard error output (STDERR). This is followed by the configuration of the console parameters, which is handled by the *ezcConsoleInput* instance. Listing 2 then creates the mandatory help option.

The next option is addressable in the shell as *--server*, or *-s* for short; it expects a string type value for the host running the IMAP server you want to query. The script additionally defines a short and an extended help text for this option and tags them as *mandatory*.

If an option is missing when the program is called, eZ Components throws an exception. This is why the help option was explicitly defined in the listing. Otherwise, you would just see an error message if you called *php mailer.php --help*.

The complete source code [3] defines three other options in the same way:

Listing 4: Retrieving Email

```
try
{
    $receiver = new ezcMailImapTransport(
        $input->getOption( "s" )->value
    );
    $receiver->authenticate(
        $input->getOption( "u" )->value,
        $input->getOption( "p" )->value
    );
    $receiver->selectMailbox( "Inbox" );
}
catch ( ezcMailException $e  )
{
    die( $output->formatText( $e->getMessage(),
 "error" ) );
}

$receiver->status( $num, $size );

$output->outputLine( "Fetching $num messages with
 a size of $size Byte.", "default", 1 );

$rawMails = $receiver->fetchByFlag( "UNDELETED"
 );
$parser = new ezcMailParser();
$mails = $parser->parseMail( $rawMails );
```

- *-u/--user* for specifying the user name on the IMAP server,
- *-p/--password* for the password and
- *-t/--target* for the target email address.

One option that differs from the rest is *-v/--verbose*: it is not defined as *mandatory*, it does not expect a text-type value (rather, an integer), and it assumes a default value of *1* if the user fails to supply a value.

Listing 3 processes the command-line parameters for the script. The script starts by registering the parameters that have been generated in the *ezcConsoleInput* object before going on to call the *process()* method. The input object throws an exception if the user passes in invalid flags. The script catches the error and outputs the error text before quitting the program. This is the first appearance for the output object defined in Listing 2. The text for this exception is formatted as an *error $output->outputLine(...)* and output on STDERR.

After successfully processing the options, the sample program checks to see whether *--help* is set, and if so, the program outputs the help text created by *ezcConsoleInput* (*$input->getHelpText()*). Finally, Listing 3 sets the *verbosityLevel* for the *ezcConsoleOutput* object.

After the *ConsoleTools* component has provided the required credentials, the sample script can now retrieve email from the IMAP server. To retrieve email, the script first instantiates the *ezcMailImapTransport* class in Listing 4; the class constructor expects the hostname of the IMAP server. A similar class exists for retrieving mail from POP mailboxes. It then goes on to authenticate the script against the server and selects the *INBOX*.

The *$receiver->status()* call returns the number of messages in the selected mailbox and the total size. The program displays both these values to the user, unless the *--verbose* parameter is set to a value of *0*. The *verbosityLevel* is passed in to *$output->outputLine()* at the point where the script displays the message. As verbosity is the third parameter for *outputLine()*, you have to specify the standard format *default*.

After this output, the IMAP transporter object creates a kind of stream with all the mail tagged with the IMAP *UNDELETED* tag. An *ezcMailParser* type object creates *ezcMail* type objects from them. A call to *$parser->parseMail()* returns the array of *ezcMail-* objects, which are processed by the code in Listing 5.

Listing 5 starts by creating a new *ezcMail* object and passing in the required data, such as the sender, recipient, and subject line. *ezcMailAddress* type objects can optionally accept the recipient's name. When cast to a string type, these objects generate an email address notation that complies with the relevant RFC. The email body is handled by an *ezcMailMultipartDigest* type object, which represents an email digest. The following *foreach* loop adds the received emails to the digest.

The program then creates another transporter object, which sends the mail. To do so, *ezcMailMtaTransport* calls the built-in PHP *mail()* function. *ezcMailSmtpTransport* would set up a socket to an SMTP server for this. Another status message with a *verbosityLevel* of *1* confirms that the send action was successful. Finally, the script deletes the messages it has processed from the server. To do so, it queries the email iterator and calls the IMAP object's *delete()* method for each mail. It also outputs a status message to indicate that all the messages have been deleted.

Conclusions

eZ Components is a component library with a standardized and well-documented API. The variety of components accelerates the development process and ensures quality. The package is available under a free license and can be sold with almost any project, including commercial ones. The project welcomes contributions and requests, and if you get stuck, the contributor community can help via the developer mailing list. ■

INFO

[1] eZ Components *http://ez.no/ezcomponents*

[2] PHP Unit: *http://www.phpunit.de*

[3] Listings online: *http://www.linux-magazine.com/Magazine/Downloads/87/eZcomponents*

Listing 5: Packing and Sending Mail

```
$forwardMail = new ezcMail();
$forwardMail->from = new ezcMailAddress(
 "do-not-reply@is-geek.de" );
$forwardMail->addTo( new ezcMailAddress(
 $input->getOption( "t" )->value ) );
$forwardMail->subject = "Mail from " . date( "Y/
 m/d H:i" );
$forwardMail->body = new ezcMailMultipartDigest()
 ;

foreach( $mails as $mail )
{
    $forwardMail->body->appendPart( new
 ezcMailRfc822Digest( $mail ) );
}

$sender = new ezcMailMtaTransport();
$sender->send( $forwardMail );

$output->outputLine( "Sent $num emails to
 {$input->getOption( "target" )->value}.",
 "default", 1 );

foreach ( $rawMails->getMessageNumbers() as $no )
{
    $receiver->delete( $no );
}

$receiver->expunge();

$output->outputLine( "Deleted $num emails.",
 "default", 1 );
```