

Welcome!

eZ components

Welcome!

Derick Rethans - dr@ez.no

<http://files.derickrethans.nl/ezc-intro-ez.pdf>

Library Goals

- Provide a solid platform for PHP application development
- Clean and simple API
- Keep backward compatibility for longer periods of time
- Stable and few regressions
- Clean IP, Open Source friendly

License and Contributing

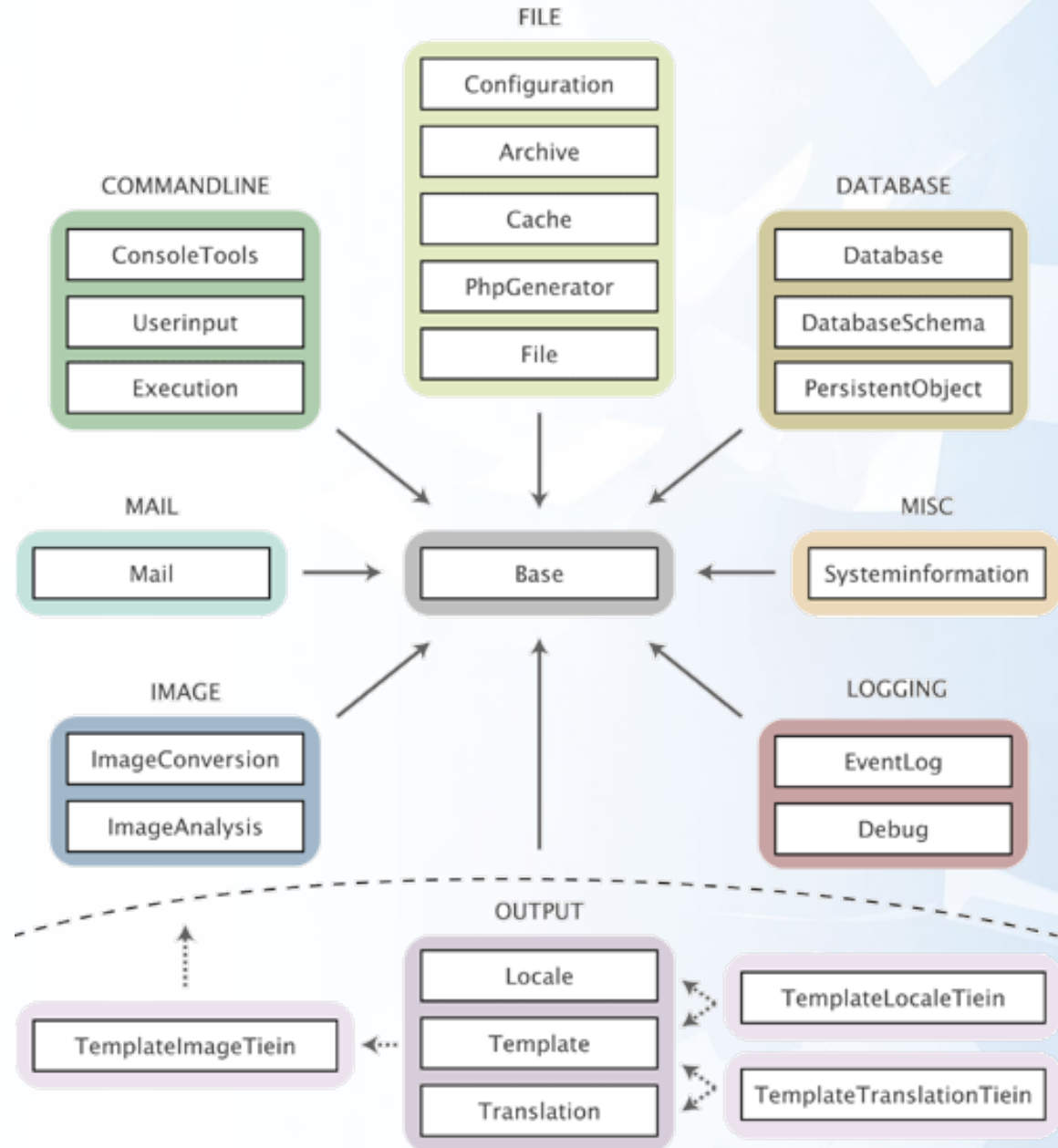
- New BSD license: Open Source, very permissive, compatible with GPL
- CLA: To accept contributions to the components we need a signed "Contributor Licensing Agreement"

Development Goals

- Move to PHP 5.1.1
- Thoroughly plan the product to get a top notch API for the library
- Thoroughly tested. All code should be covered by unit tests prior to implementation (PHP Unit)
- Make sure we keep the product open enough for future development
- Do proper documentation during the development

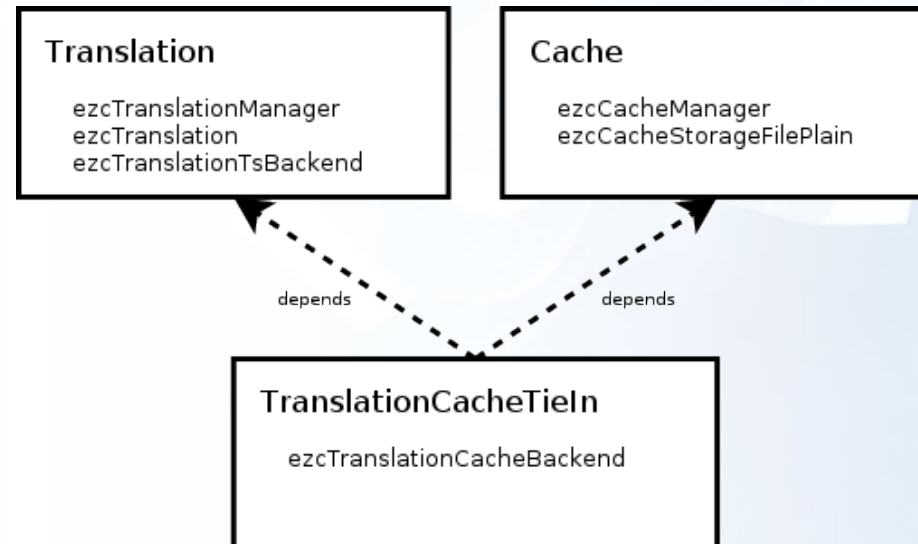
Overview

Share your information



Dependencies

- The less the better
- Only if really necessary
- Dependency-only packages
- Tie-Ins



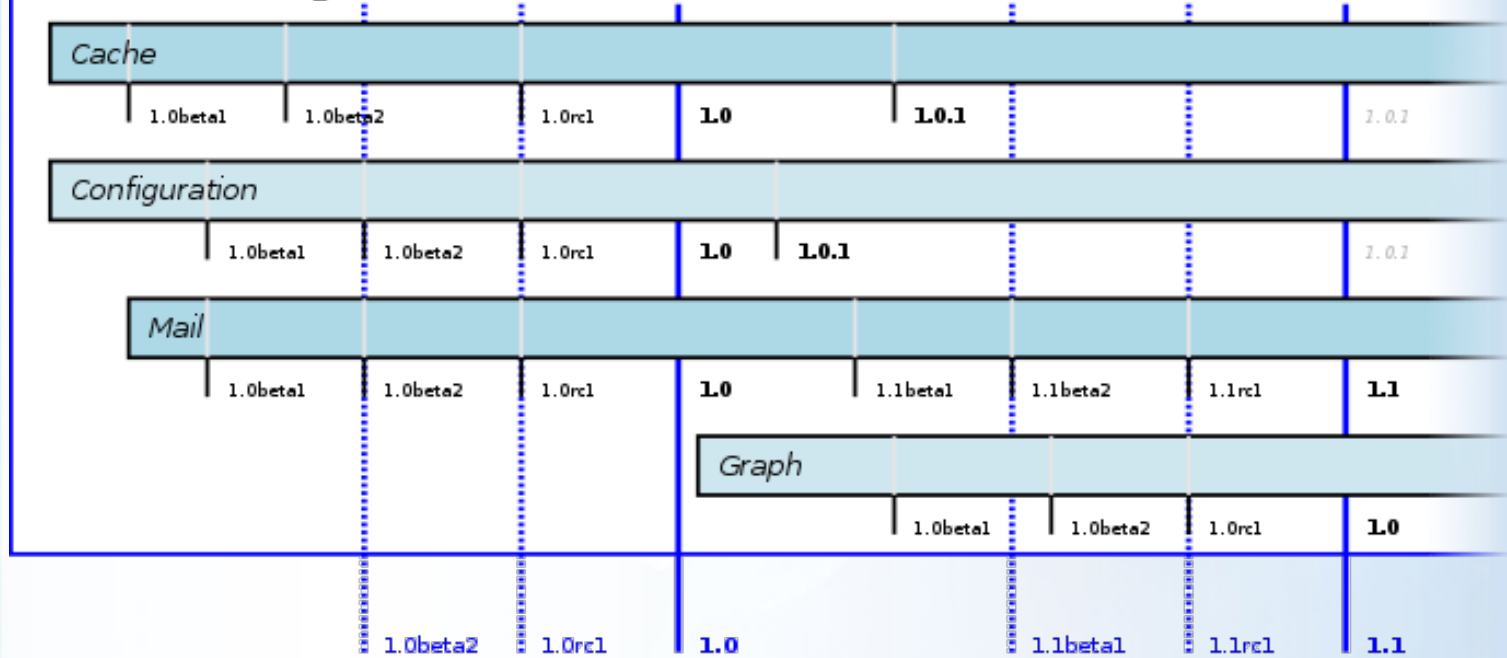
Component Versioning

Versions for components have three elements: x.y.z, which have the following meaning:

- x: major version number. A component with major version 0 can never be released publically (beta). It will only increase when there is a backwards compatible break in the component's API.
- y: minor version number, is used for all feature additions.
- z: mini version number, is used to denote bugfixes only. This third part can also be a string in the set: (alpha, beta1, beta2, betaN, rc1, rc2, rcN).

x and y show the version number of the component, the z is an addition showing the state (beta etc) or which bugfix release it is.

eZ components



Documentation:

- Makes a library usable
- API documentation
- Tutorials
- Example Applications
- <http://ez.no/doc/components/overview>

PHP Documentor:

- Made for PHP
- In use by PEAR
- Supported by PHP IDEs

Installation:

```
// download the bundle from http://ez.no/download/ez\_components  
tar -xjf ezcomponents-1.1beta1.tar.bz2  
pwd
```

Setup:

```
<?php  
ini_set( 'include_path', '/home/httpd/ezcomponents/trunk..' );  
require 'Base/src/base.php';  
  
function __autoload( $className )  
{  
    ezcBase::autoload( $className );  
}  
  
ezcBase::checkDependency( 'Test', ezcBase::DEP_PHP_VERSION, '5.3.0' );  
?>
```

PEAR Channels

- Allows external entities to setup "PEAR" Servers
- Dependency Validation
- Aggregated Into <http://pearaside.net>

A channel defines:

- The channel name.
- An optional suggested user alias for the channel.
- A brief summary of the channel's purpose.
- An optional package to perform custom validation of packages on both download and packaging.
- A list of protocols supported by a channel (XML-RPC, SOAP, and REST are supported).
- A list of mirrors and the protocols they support.

Installation:

```
pear channel-discover components.ez.no
pear remote-list -c ezc
pear install ezc/ezcomponents
```

Setup:

```
<?php
require 'ezc/Base/base.php';

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

ezcBase::checkDependency( 'Test', ezcBase::DEP_PHP_VERSION, '5.3.0' );
?>
```

What's Now?

- Archive: tar (gzip/bzip2) and zip reading and writing
- Base: base for all components
- Cache: Caching of data
- Configuration: Reading/Writing .ini files
- ConsoleTools: Tools for writing console scripts
- Database: Database and SQL abstraction layer
- DatabaseSchema: Database schema manipulation
- Debug: Debugging utilities
- EventLog: Logging application events
- EventLogDatabaseTiein: Database backend for EventLog

What's Now?

- Execution: Handling script execution abortion
- File: Common file operations
- ImageAnalysis: Analysing image data (Exif, size)
- ImageConversion: Manipulating images
- Mail: Mail creating, sending, receiving and parsing
- PersistentObject: ORM database layer
- PersistentObjectDatabaseSchemaTiein: Scripts for generation PersistentObject definitions from database schemas
- PhpGenerator: Generating PHP files
- SystemInformation: Obtaining information about a system (CPU, Memory...)

What's Now?

- Template: Template processing
- Translation: Providing translatable string support
- TranslationCacheTiein: Caching support for Translation
- UserInput: Form handling and input filtering

What's Next?

New components:

- Authentication
- ContentDiff
- Feed: RSS/ATOM feeds
- Graph: Graphs and diagrams
- SignalObserver: IPC for PHP applications
- URL: Obtaining parameters from URLs and creating them

Additions to components:

- Database: Oracle support
- DatabaseSchema: Support for SQLite, PostgreSQL and Oracle
- Mail: More authentication mechanisms
- PersistentObject: Class relations
- Template: Custom functions and character set support

Questions and Resources

Questions anybody?

Resources:

- Download: http://ez.no/download/ez_components
- Documentation: <http://ez.no/doc/components/overview>
- Mailinglist: <http://lists.ez.no/mailman/listinfo/components>
- These Slides: <http://files.derickrethans.nl/ezc-intro-ez.pdf>

Repository Layout

Main directories:

- docs/: Contains articles, some status lists and development guidelines.
- release-info/: For each version a file describing which package and version is part of that release.
- releases/: releases/[packagename]/[version] is a copy of trunk/[packagename] at the time a component was released.
- scripts/: Maintenance scripts.
- trunk/: Code in development.

Package Layout

trunk/[packagename]/:

- ChangeLog: Contains the changes for this package.
- CREDITS, DESCRIPTION: Package credits and description.
- DEPS: A list of packages and versions this package depends on.
- design/: Design and requirement documents.
- docs/: Tutorial, examples and other documentation.
- src/: The source code of a package.
- tests/: Unit tests.

trunk/[packagename]/src/:

- exceptions/: Exception classes.
- interfaces/: Interfaces and abstract classes.
- options/: Contains option classes (extend from ezcBaseOptions class).
- structs/: "Struct" classes.
- [dir]/: Classes grouped logically.
- [package]_autoload.php: Autoload definition.
- [file].php: "Main" classes.

Tests Layout

trunk/[packagename]/tests/:

- suite.php: Contains the changes for this package.
- [dir]/ and [file].php: Contains the test cases, often in the same structure as the source.

Installation:

```
// checkout trunk and script directories:
mkdir ezcomponents
cd ezcomponents
svn co http://svn.ez.no/svn/ezcomponents/scripts
svn co http://svn.ez.no/svn/ezcomponents/trunk

// Setup autoload environment:
scripts/setup-env.sh
```

Setup:

```
<?php
ini_set( 'include_path', '/home/httpd/ezcomponents/trunk..' );
require 'Base/src/base.php';

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

ezcBase::checkDependency( 'Test', ezcBase::DEP_PHP_VERSION, '5.3.0' );
?>
```

Classnames

- Pre-fixed: for namespacing
- Readable: for sanity
- Slightly Mangled: for clarity

ezcMail vs. Mail

ezcTestSuite vs. PHPUnit2_Framework_TestSuite

ezcMailSmtptTransport vs. ezcMailSMTPTransport

Autoload Arrays

Using the classname's part as path elements makes ugly paths:

```
ezcMailTransportMta           => mail/transport/mta.php
ezcMailTransportSmt           => mail/transport/smt.php
ezcMailException              => mail/exception.php
ezcMailTransportSmtException => mail/transport/smt/exception.php
```

More logical names (mail_autoload.php):

```
ezcMailTransportMta           => transports/transport_mta.php
ezcMailTransportSmt           => transports/transport_smt.php
ezcMailException              => exceptions/mail_exception.php
ezcMailTransportSmtException => exceptions/transport_smt_exception.php
```

Some problems:

- Clashes in first part of the classname
- Needs installation into correct place for development

```
<?php
require 'ezc-setup.php';

$cfg = ezcConfigurationManager::getInstance();
$cfg->init( 'ezcConfigurationIniReader', dirname( __FILE__ ). '/cfg' );

$pw = $cfg->getSetting( 'example', 'db', 'password' );
echo "The password is '$pw'.\n";
?>
```

```
<?php
require 'ezc-setup.php';

$reader = new ezcConfigurationIniReader();
$reader->init( dirname( __FILE__ ). '/cfg', 'example2' );
// Validation
$result = $reader->validate();
foreach ( $result->getResultList() as $resultItem )
{
    echo htmlspecialchars( basename( $resultItem->file ) . ":" .
        $resultItem->line . ":" . $resultItem->column . ":" . " " .
        $resultItem->details . "\n" );
}

// Reading
$reader->init( dirname( __FILE__ ). '/cfg', 'example' );
$config = $reader->load();

// Writing
$writer = new ezcConfigurationArrayWriter();
$writer->init( dirname( __FILE__ ). '/cfg', 'example', $config );
$writer->save();
?>
```

Archive

```
<?php
require 'ezc-setup.php';

$archive = ezcArchive::open( dirname( __FILE__ ) . '/test-archive.zip' );
foreach( $archive as $entry )
{
    if ( $entry->getPath() == 'archive-logo.jpg' )
    {
        $dir = dirname( __FILE__ );
        echo "Extracting {$entry->getPath()} to $dir.<br/><br/>\n";
        $archive->extractCurrent( $dir );
    }

    echo $entry, "\n";
}
?>
```



```
<?php
require 'ezc-setup.php'; @mkdir( '/tmp/temp-location' );

ezcCacheManager::createCache(
    'identifier', '/tmp/temp-location',
    'ezcCacheStorageFileEvalArray', array( 'ttl' => 30 ) );
$cache = ezcCacheManager::getCache( 'identifier' );
$myId = 'unique_id_1';

if ( ( $data = $cache->restore( $myId ) ) === false )
{
    $data = "Plain cache stored on " . date( 'Y-m-d, H:i:s' );
    $cache->store( $myId, $data );
}
echo $data;
```

- FilePlain: Raw data
- FileArray: var_export() data, include to restore
- FileEvalArray: var_export() data, eval() to restore

```

<?php
require 'ezc-setup.php';
$out = new ezcConsoleOutput;
$out->formats->headline->color = 'red';
$out->formats->headline->style = array( 'bold' );

$table = new ezcConsoleTable( $out, 60 );
$table[0]->format = 'headline';
$table[0]->borderFormat = 'headline';

$table[0][0]->content = 'Headline 1';
$table[0][1]->content = 'Headline 2';
$table[1][0]->content = "The next cell will wrap:";
$table[1][1]->content = <<<END
If there is a lot of text in a specific cell then the
ezcConsoleTable will correctly wrap around.
END;
$table[1][1]->align = ezcConsoleTable::ALIGN_RIGHT;

$table->outputTable();
?>

```

```

+-----+-----+
| Headline 1 | Headline 2 |
+-----+-----+
| The next cell will wrap: | If there is a lot of text in a | |
| | | specific cell then the |
| | | ezcConsoleTable will correctly |
| | | wrap around. |
+-----+-----+

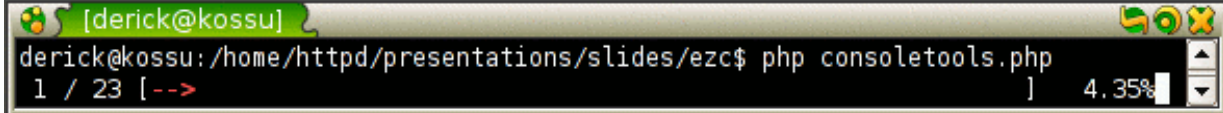
```

```
<?php
require 'ezc-setup.php';

$max = 23;
$output = new ezcConsoleOutput;
$output->formats->bar->color = 'red';
$output->formats->bar->style = array( 'bold' );
$options = array(
    'emptyChar'    => ' ',
    'barChar'      => '-',
    'formatString' =>
        '%act% / %max% [' .
        $output->formatText( '%bar%', 'bar' ).
        '] %fraction%%',
);
$progress = new ezcConsoleProgressbar( $output, $max, $options );

for ( $i = 0; $i < $max; $i++ )
{
    usleep( 20000 );
    $progress->advance();
}

$progress->finish();
?>
```



```
[derick@kossu]
derick@kossu: /home/httpd/presentations/slides/ezc$ php consoletools.php
1 / 23 [ --> ] 4.35%
```

```
<?php
require 'ezc-setup.php';
$optionHandler = new ezcConsoleInput();

$file = new ezcConsoleOption(
    'f', 'file', ezcConsoleInput::TYPE_STRING
);
$file->mandatory = true;
$optionHandler->registerOption( $file );

try
{
    $optionHandler->process();
}
catch ( ezcConsoleException $e )
{
    die( $e->getMessage() );
}

echo "Processing file: ",
    $optionHandler->getOption( 'f' )->value, "\n";
?>
```

```
<?php
require 'ezc-setup.php';

class myExecutionHandler extends ezcExecutionBasicErrorHandler
{
    public static function onError( Exception $exception = NULL )
    {
        echo '<div class="shadow" style="margin: 1em 4em 0.8em 3em;"><div
class="output" style="font-size: 1.8em; margin: -0.555555555555556em 0 0
-0.555555555555556em; background: #eeee33;">';
        parent::onError( $exception );
    }
}

ezcExecution::init( 'myExecutionHandler' );

//ezcExecution::cleanExit();
?>
```

Supported Handlers:

- MySQL
- PostgreSQL
- Oracle
- SQLite

Using Instances:

```
<?php
require 'ezc-setup.php';

$db1 = ezcdbFactory::create( 'mysql://root@localhost/ezc' );
ezcdbInstance::set( $db1, 'ezc' );

$strunk = ezcdbFactory::create( 'mysql://root@localhost/eztrunk' );
ezcdbInstance::set( $strunk, 'ezt' );

$db = ezcdbInstance::get( 'ezt' );
echo $db->getName();
?>
```

```
<?php
require 'ezc-setup.php';
$db = ezcdbFactory::create( 'mysql://root@localhost/geolocation' );
$sq = $db->createSelectQuery();

$stmt = $sq->select( 'name', 'country', 'lat', 'lon' )
    ->from( 'city' )
    ->where(
        $sq->expr->like(
            'normalized_name', $sq->bindValue( 'orlando%' )
        )
    )
    ->orderBy( 'country', 'name' )
    ->prepare();
$stmt->execute();

foreach ( $stmt as $entry )
{
    list( $name, $country, $lat, $lon ) = $entry;
    printf( '%s, %s is @ %.2f%s %.2f%s<br/>',
        $name, $country,
        abs( $lat ), ( $lat > 0 ? "N" : "S" ),
        abs( $lon ), ( $lon > 0 ? "E" : "W" ) );
}
?>
```

```
<?php
require 'ezc-setup.php';
$session = new ezcPersistentSession(
    ezcDbInstance::get(),
    new ezcPersistentCodeManager( "path/to/definitions" )
);

// Creating New Objects
$object = new City();
$object->normalized_name = "dieren";
$object->name = 'Dieren';
$object->country = 'NL';
$session->save( $object );

// Finding Objects
$q = $session->createFindQuery( 'City' );
$q->where(
    $sq->expr->like( 'name', $sq->bindValue( 'oslo%' ) )
)
->orderBy( 'country', 'name' )
->limit( 10 );
$objects = $session->findIterator( $q, 'City' );

foreach ( $objects as $object )
{
    // ...
}
?>
```

```
<?php
require 'ezc-setup.php';

// Create the schema objects
$db = ezcDbFactory::create( 'mysql://root@localhost/geolocation' );
$dbSchema = ezcDbSchema::createFromDb( $db );
$wantedSchema = ezcDbSchema::createFromFile(
    'xml', dirname( __FILE__ ) . '/geoschema.xml'
);

// Compare the schemas
$differences = ezcDbSchemaComparator::compareSchemas(
    $dbSchema, $wantedSchema
);

// Show the differences
foreach( $differences->convertToDDL( $db ) as $query )
{
    echo $query, "<br/>\n";
}
?>
```

```
<?php
require 'ezc-setup.php';
$log = ezcLog::getInstance();
$log->source = $log->category = NULL;

// Create writers
$warningWriter = new ezcLogUnixFileWriter(
    "/tmp", "ezc-pres-warning.log", 128
);
$errorWriter = new ezcLogUnixFileWriter(
    "/tmp", "ezc-pres-error.log", 256
);

// Create filters
$warningFilter = new ezcLogFilter;
$warningFilter->severity = ezcLog::WARNING;
$log->getmapper()->appendRule(
    new ezcLogFilterRule( $warningFilter, $warningWriter, true )
);

$errorFilter = new ezcLogFilter;
$errorFilter->severity = ezcLog::ERROR;
$log->getmapper()->appendRule(
    new ezcLogFilterRule( $errorFilter, $errorWriter, true )
);

// Log messages
$log->log( "Oops, this was unexpected.", ezcLog::WARNING );
$log->log( "Oh no, major problem!", ezcLog::ERROR, array(
    "message" => "SQL" )
);
```

```
<?php
require 'ezc-setup.php';
$dir = dirname( __FILE__ );

$mail = new ezcMailComposer();
$mail->from = new ezcMailAddress( 'john@doe.com', 'John Doe' );
$mail->addTo( new ezcMailAddress( 'dr@ez.no', 'Derick Rethans' ) );

$mail->subject = "Example of an HTML email with attachments";
$mail->plainText = "Here is the text version of the mail.
This is displayed if the client can not understand HTML";

$mail->htmlText = <<<ENDHTML
<html>
Here is the HTML version of your mail
with an image: <img src='file://$dir/consoletools-table.png' />
</html>
ENDHTML;

$mail->addAttachment( "$dir/mail.php" );
$mail->build();

echo "<pre><font size='4'>",
    htmlspecialchars(
        $mail->generateHeaders() . "\r\n" . $mail->generateBody() );

$transport = new ezcMailTransportSmtplib( 'localhost', null, null, 2525 );
//$transport->send( $mail );
?>
```

```
<pre><font size="4"><?php
require 'ezc-setup.php';

$mailbox = new ezcMailMboxTransport( dirname( __FILE__ ) . "/mbox-
example.mbox" );
$set = $mailbox->fetchAll();
$parser = new ezcMailParser();
$mail = $parser->parseMail( $set );
showParts( $mail[0], 0 );

function showParts( $mail, $level )
{
    switch ( get_class( $mail ) )
    {
        case 'ezcMail':
            printf( "%s%s\n", str_repeat( ' ', $level ),
                'Mail' );
            showParts( $mail->body, $level + 1);
            break;

        case 'ezcMailText':
            printf( "%s%s (%s, %s)\n", str_repeat( ' ', $level ),
                'Text', $mail->subType, $mail->charset );
            echo "---\n", htmlspecialchars( $mail->text ), "\n---\n";
            break;

        case 'ezcMailFile':
            printf( "%s%s (%s)\n", str_repeat( ' ', $level ),
                'File', $mail->fileName );
            break;
    }
}
```

```
<?php
require 'ezc-setup.php';

$image = new ezcImageAnalyzer( dirname( __FILE__ ) . '/moon.jpg' );

if ( in_array( $image->mime, array( 'image/tiff', 'image/jpeg' ) ) )
{
    $date = date( DATE_RFC822, $image->data->date );
    $exif = $image->data->exif['EXIF'];

    $shutterSpeedArray = split( '/', $exif['ExposureTime'] );
    $shutterSpeed = "1/" . $shutterSpeedArray[1] / $shutterSpeedArray[0] . "
sec";

    $apertureArray = split( '/', $exif['FNumber'] );
    $aperture = "F" . round( $apertureArray[0] / $apertureArray[1], 1 );

    $focalLengthArray = split( "/", $exif['FocalLength'] );
    $focalLength = round( $focalLengthArray[0] / $focalLengthArray[1], 2 ) .
" mm";

    print <<<OUTPUTEND
<dl>
    <di>Photo taken on:</di><dd>$date</dd>
    <di>Shutter speed:</di><dd>$shutterSpeed</dd>
    <di>Aperture:</di><dd>$aperture</dd>
    <di>Focal length:</di><dd>$focalLength</dd>
</dl>
OUTPUTEND;
}
```

```
<?php
require 'ezc-setup.php';
$dir = dirname( __FILE__ );
// Setup
$filters = array();
$settings = new ezcImageConverterSettings(
array( new ezcImageHandlerSettings( 'GD', 'ezcImageGdHandler' ) )
);
$converter = new ezcImageConverter( $settings );

// Create transformations
$filters[] = new ezcImageFilter( 'scale',
array(
    'width' => 320, 'height' => 240,
    'direction' => ezcImageGeometryFilters::SCALE_DOWN,
)
);
$converter->createTransformation(
'preview', $filters, array( 'image/jpeg' )
);

$filters[] = new ezcImageFilter( 'colorspace',
array( 'space' => ezcImageColorspaceFilters::COLORSPACE_GREY )
);
$converter->createTransformation(
'prevgrey', $filters, array( 'image/jpeg' )
);
?>
```

ImageConversion

```
<?php
require 'imageconversion.php';

// Apply transformations
$converter->transform(
    'preview', "$dir/nacreous.jpg", "$dir/nacreous-small.jpg"
);
$converter->transform(
    'prevgrey', "$dir/nacreous.jpg", "$dir/nacreous-grey.jpg"
);
?>
```

Result:



Development Process

- Suggest the idea on the components list
- Collect feedback and wait until the idea is approved
- Write requirements and design document
- Announce those docs and ask for comments on the components list
- Repeat step 3 to 4 until everybody is happy
- Implement class stubs including API docs
- For each method do step 8 and 9
- Write testcase for method
- Write code for method and run test case
- In case API needs to be changed, discuss that on the list and repeat step 6 to 9
- Make the class documentation more extensive and add some class level examples
- Write a tutorial
- Ask for review again

Questions and Resources

Questions anybody?

Resources:

- Download: http://ez.no/download/ez_components
- Documentation: <http://ez.no/doc/components/overview>
- Mailinglist: <http://lists.ez.no/mailman/listinfo/components>
- These Slides: <http://files.derickrethans.nl/ezc-intro-ez.pdf>