

Welcome!

# About Me

- eZ systems A.S.
- Content Management: eZ publish
- Components Team

# About This Talk

- Library Architecture
- Problems
- Some Useful Packages

# 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

# Development Goals

- Move to PHP 5
- 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.
- Make sure we keep the product open enough for future development
- Do proper documentation during the development.

# What About PEAR?

PEAR is short for "PHP Extension and Application Repository" and is pronounced just like the fruit. The purpose of PEAR is to provide:

- A structured library of open-sourced code for PHP users
- A system for code distribution and package maintenance
- A standard style for code written in PHP, specified [here](#)

# Problematic PEARs

- Categorization
- Interesting Naming
- No common API or naming

# Rotten PEARs

- Not well tested
- Never finished
- Unfixed bugs
- Abandoned packages
- No documentation



# Package Definition

- Package Description
- Version Information
- Files, and Install Paths
- Dependencies

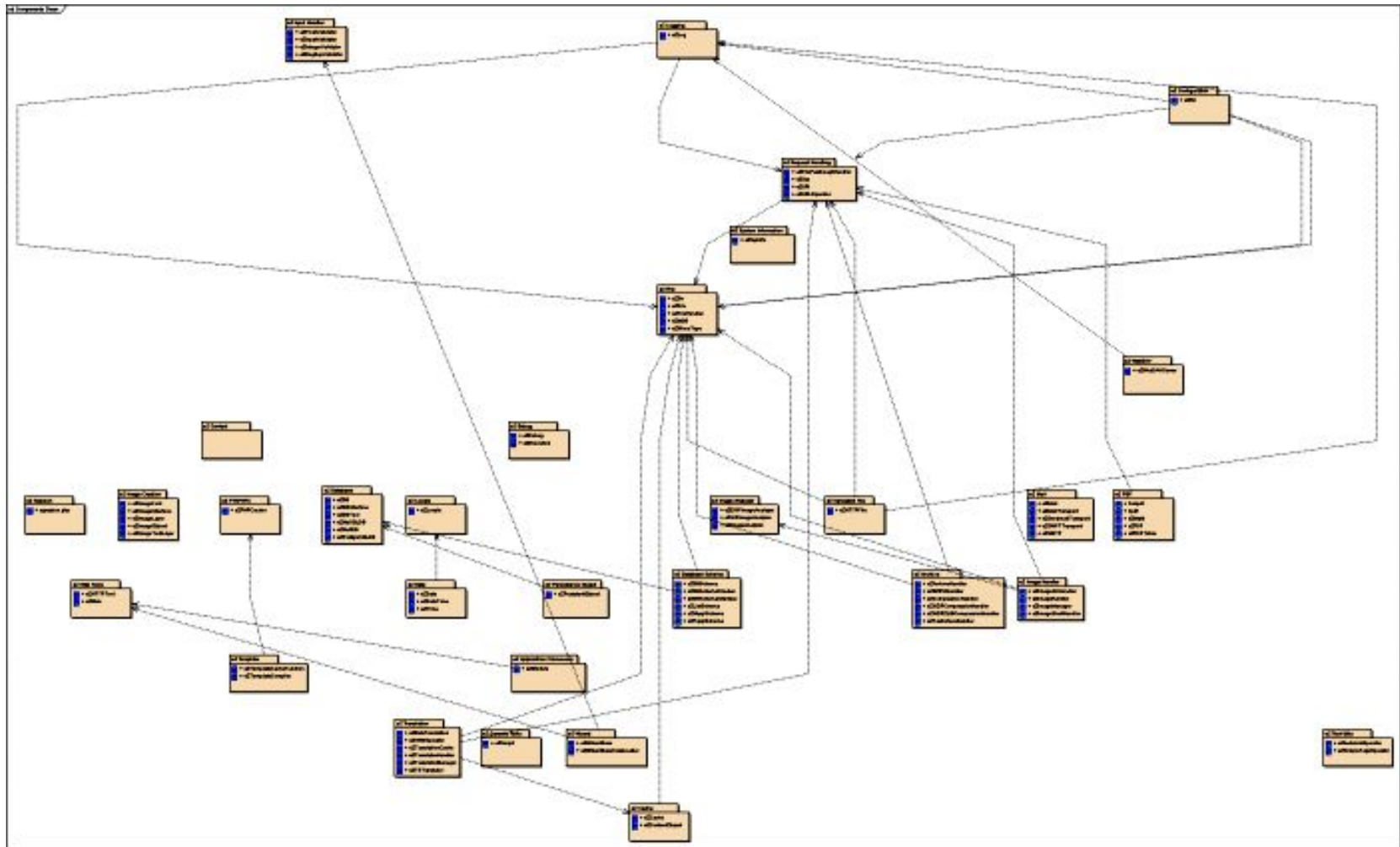
# PEAR Channels

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

A channel defines:

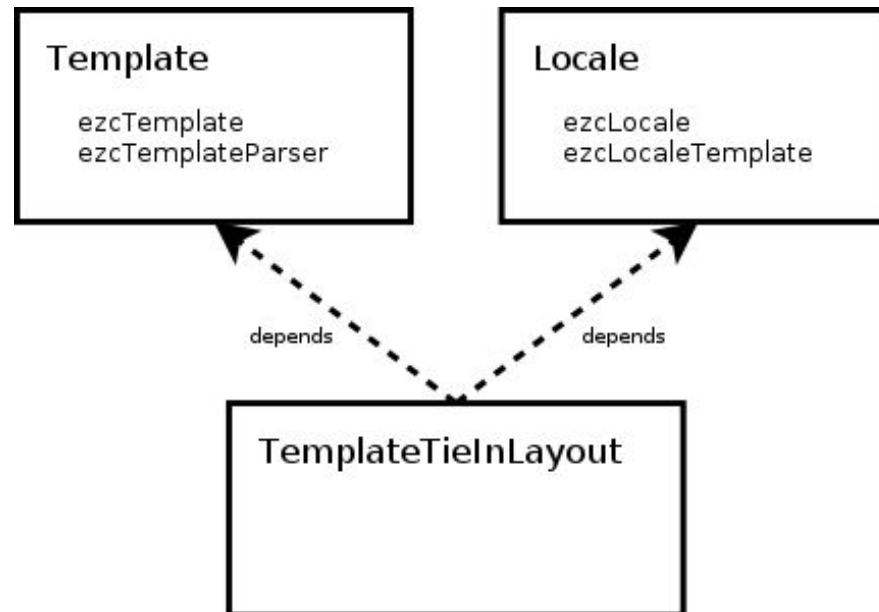
- The channel name.
- An optional suggested user alias for the channel.
- A brief summary of the channel's purpose.
- An optional package to perform custom validation of packages on both download and packaging.
- A list of protocols supported by a channel (XML-RPC, SOAP, and REST are supported).
- A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

# Original Dependencies

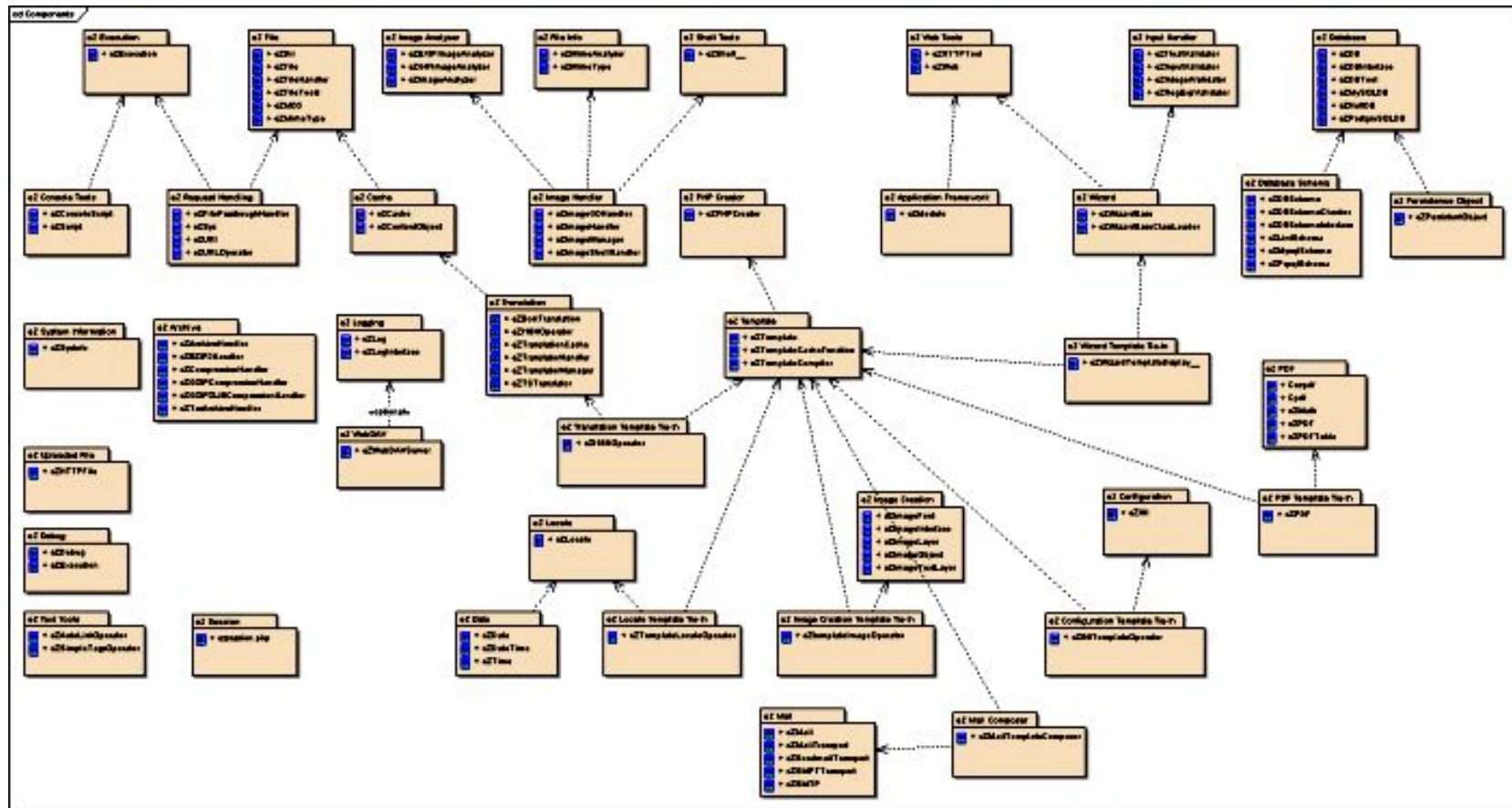


# Dependencies

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



# New Dependencies



# Classnames

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

ezcMail vs. Mail

ezcTestSuite vs. PHPUnit2\_Framework\_TestSuite

ezcMailSmtptTransport vs. ezcMailSMTPTransport

# Autoload

```
<?php
function __autoload( $className )
{
    $path = preg_replace( '/([A-Z])/', '\\1', $className );
    $path = strtolower( $path );
    $path = str_replace( 'ezc/', '', $path ) . '.php';
    echo "Loading $path...\n";
    @include_once $path;
}

$a = new ezcMailSmtptTransport;
?>
```

# Autoload Arrays

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

```
ezcMailTransportMta => mail/transport/mta.php  
ezcMailTransportSmtplib => mail/transport/smtplib.php  
ezcMailException => mail/exception.php  
ezcMailTransportSmtplibException => mail/transport/smtplib/exception.php
```

More logical names (mail\_autoload.php):

```
ezcMailTransportMta => transports/transport_mta.php  
ezcMailTransportSmtplib => transports/transport_smtplib.php  
ezcMailException => exceptions/mail_exception.php  
ezcMailTransportSmtplibException => exceptions/transport_smtplib_exception.php
```

Some problems:

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



# Autoload and Exceptions

```
<?php
    try
    {
    }
    catch( ezcA $e )
    {
    }
?>
```

This would always call the `__autoload()` function to load the exception class, even if it's not needed.

# Documentation

## Documentation:

- Makes a library usable
- API documentation
- Examples
- Test Cases

## PHP Documentator:

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

# Configuration

## Current:

```
<?php
class eZTranslationCache {
    function cacheDirectory() {
        require_once 'lib/ezutils/classes/ezini.php';
        $ini =& eZINI::instance();
        $locale = $ini->variable( 'RegionalSettings', 'Locale' );
    }
}
?>
```

That would make almost all components depend on *eZINI*, and that's something we want to avoid.

## New:

```
<?php
class ezcTranslationCache {
    protected $region;

    function __constructor( $region ) {
        $this->region = $region;
    }
}
```

# Configuration

When there are a lot of options:

- Use constructor for most used ones
- Provide defaults for all others
- Add a *setOptions()* function that accepts an array with options

# Debugging

## Current:

```
<?php
function timestamp( $name )
{
    if ( !isset( $this->Timestamps[$name] ) ) {
        eZDebug::writeError( "...", '...::timestamp' );
        return false;
    }
    return $this->Timestamps[$name];
}
?>
```

That would make almost all components depend on *eZDebug*, and that's something we want to avoid too.

## New:

```
<?php
class ezcBase {
    function debug() { }
}

set_error_handler(array('ezcBase', 'debug'));
```

# 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, beta $N$ , rc1, rc2, rc $N$ ).

$x$  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.

# Component Versioning

- *0.1.0*: first non-publically released version
- *1.0.0*: first publically released version
- *1.0.1*: first bugfix release for component version "1.0"
- *1.3.7*: 7th bug fix release for version "1.3" - version "1.3" has more features compared to "1.0"-*"1.2"*, but does not break BC
- *1.4beta1*: First beta release for "1.4".
- *1.4rc1*: First release candidate for "1.4".
- *1.4.0*: First production ready release of "1.4".
- *2.0alpha*: Development release, where backward compability is broken compared to version "1.x".
- *2.0beta1*: First beta release of "2.0".
- *2.0rc1*: First release candidate "2.0".
- *2.0.0*: First production ready release "2.0".

# E\_RECOVERABLE\_ERROR

```
<?php
class Foo {
}

function blah (Foo $a)
{
}

function error()
{
$a = func_get_args();
var_dump($a);
}

set_error_handler('error');

blah (new StdClass);
echo "ALIVE!\n";
?>
```

- Used if there is a fatal error which doesn't leave the engine in an unstable state.
- If not handled in a user defined error handler -> E\_FATAL.



# What is Wrong Here?

```
<?php
    $sql = "
        SELECT card_num, card_name, card_expiry
        FROM credit_cards
        WHERE uid = '{$_GET['uid']}'
    ";
?>
```

<http://example.com/script.php?uid=42>

```
SELECT card_num, card_name, card_expiry
FROM credit_cards
WHERE uid = '42'
```

!

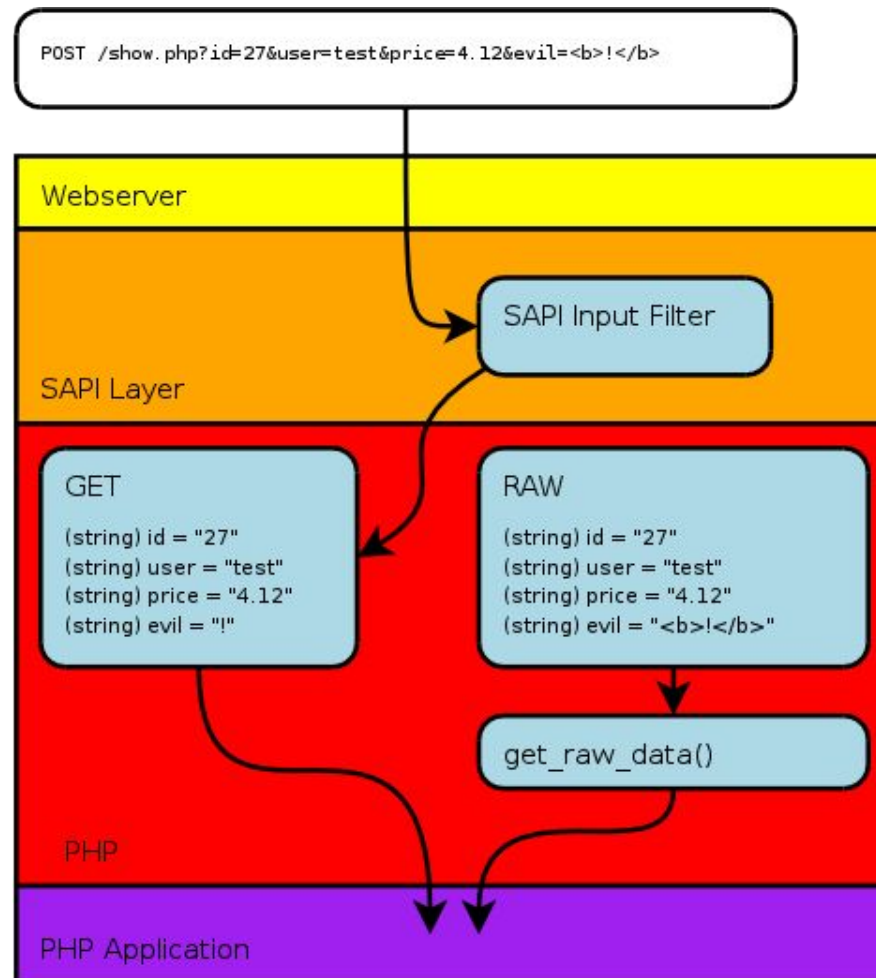
<http://example.com/script.php?uid=42'%20or%20''='>

```
SELECT card_num, card_name, card_expiry
FROM credit_cards
WHERE uid = '42' or ''=''
```

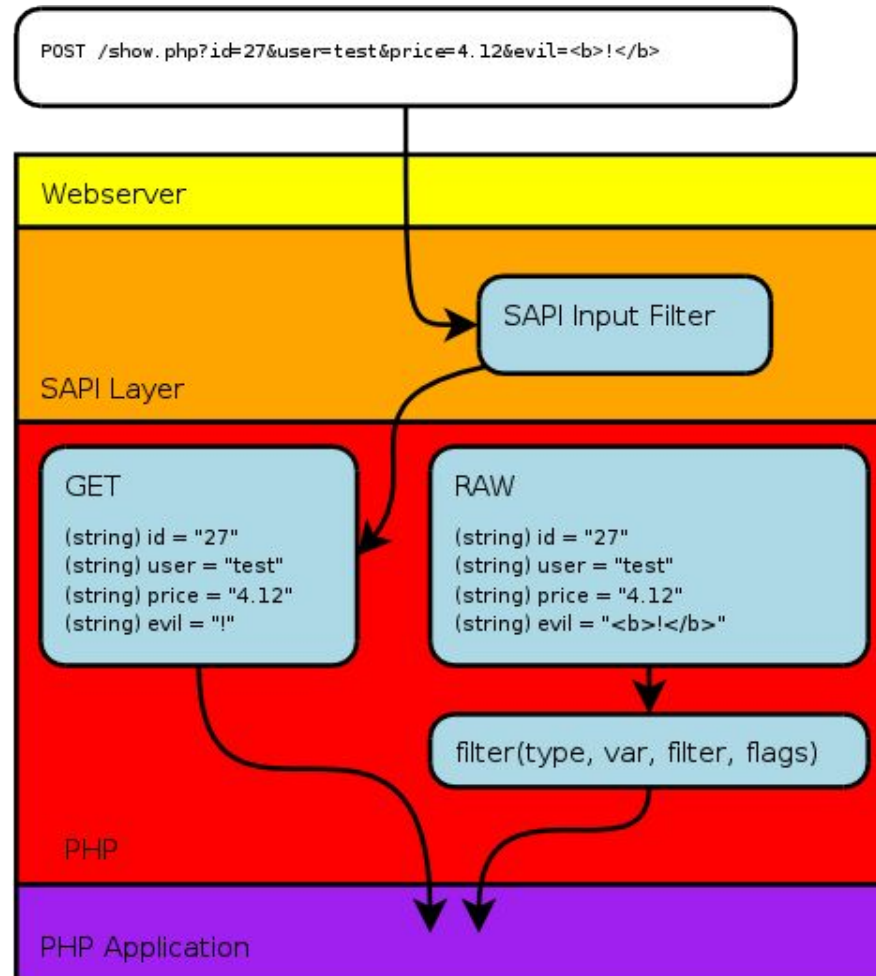
# SAPI Input filter

- Sits between PHP and the webserver
- Is used while fetching data from users sources
- Can be used to filter data
- Prohibit data from entering PHP
- Written as a C extension to PHP
- Server wide filter

# First Idea of an Input Filter



# Second Idea of an Input Filter



# Input Filter

```
mixed input_get (int source, string name, string filter  
                [, mixed filter_options, [ string charset ] ]);
```

- Returns the filtered variable `$name` from source `$source`. It uses the filter as specified in `$filter`, and additional options to the filter through `$filter_options`.

```
bool filter_data (mixed variable, string filter  
                 [, mixed filter_options, [ string charset ] ]);
```

- Filters the user supplied variable `$variable` with the filter `$filter` using the options as specified in `$filter_options`. This function modifies the original variable, and returns false if the filter failed, or true if it was successful.

# Input Filter

- *string*: Returns the input variable stripped of XML/HTML tags and other things that can cause XSS problems.
- *encoded*: Encodes all 'special' characters as URL encoded values.
- *special\_chars*: Encodes all 'special' characters as XML entities.
- *unsafe\_raw*: Returns the input variable as-is.
- *email*: Removes all characters that can not be part of a correctly formed e-mail address.
- *url*: Removes all characters that can not be part of a correctly formed URI.
- *number\_int*: Removes everything [^0-9].
- *number\_float*: Removes everything [^0-9.].
- *magic\_quotes*: BC filter for people who like magic quotes.

# Input Filter

- *int*: Returns the data as an integer.
- *boolean*: Returns true for '1' and true and 'false' for '0' and 'false'.
- *float*: Returns the data as a floating point value.
- *regexp*: Matches the input value as a string against the regular expression. If there is a match then the string is returned, otherwise the filter returns "false".  
Remarks: Only available if pcre has been compiled into PHP.

# Transliteration

- Non-Latin scripts can't be used in a lot of cases, f.e.
- Extended-"ASCII" can also not be used
- Transliteration converts characters to other characters
- Existing extension in PECL: `pecl/translit`
- New extension based on ICU follows soon: `i18n_translit`



# Transliteration



## What we did first:

"Håtveit på 8. plass" to "h\_tveit\_p\_8\_plass"

## What we do now:

"Håtveit på 8. plass" to "haatveit\_paa\_8\_plass"

## How we do this:

```
<?php
$string = "Håtveit på 8. plass";
$res = transliterate($string,
    array('normalize_ligature', 'lowercase_latin',
        'space_to_underscores'),
    'utf-8', 'utf-8');
?>
```

# Transliteration

```
<?php
    $string = <<<END
След малко се запътвам към автобусната спирка, от там на
летището, после пак на летището и пак на автобусната спирка
и в Пловдив. Мозафока.
END;

    $string = iconv("utf-8", "ucs-2", $string);
    $res = transliterate($string, array('cyrillic_transliterate'));
    echo iconv('ucs-2', 'utf-8', $res);
?>
```

# Transliteration

```
<?php
```

```
    $string = <<<END
```

```
    美军总攻费卢杰 战况惨烈
```

```
מורדים בפלוג'ה משגרים רקטה בניסיון לעצור את הניסיון של כוחות  
ארה"ב לכבוש את העיר תצלום: אי-פי
```

```
Υποθέτω πως για τους ενασχολούντες, η είδηση της βάφτισης  
του linux kernel tree σε 2.6 στις 17 του Δεκέμβρη 2003 είναι  
ήδη γνωστή.
```

```
END;
```

```
    $string = iconv("utf-8", "ucs-2", $string);
```

```
    $res = transliterate($string, array(
```

```
        'han_transliterate', 'hebrew_transliterate',
```

```
        'greek_transliterate'));
```

```
    echo nl2br(iconv('ucs-2', 'utf-8', $res));
```

```
?>
```

# Normalization

```
<?php
    $string = <<<END
Normalization:
This example is called «normalization». It`s used to convert
"curly quotes", special-hypens and other characters to more `ascii`
representations.
```

Removing diacritical marks:

```
Another filter removes diacritical marks.
END;
```

```

    $string = iconv("utf-8", "ucs-2", $string);
    $res = transliterate($string, array(
        'normalize_punctuation', 'diacritical_remove'));
    echo nl2br(iconv('ucs-2', 'utf-8', $res));
?>
```

# Other filters

```
<pre><?php
    $string = <<<END
decompose_currency_signs: € £ ¥
decompose_special: © ± « »

normalize_numbers: . * . +

han_transliterate:
uppercase_cyrillic: След малко се запътвам
END;

$string = iconv("utf-8", "ucs-2", $string);
$res = transliterate($string, array(
    'decompose_currency_signs', 'decompose_special',
    'normalize_numbers', 'han_transliterate',
    'uppercase_cyrillic', 'normalize_ligature',
    'diacritical_remove'));
echo iconv('ucs-2', 'utf-8', $res);
?>
```

# Problems

One identifier can mean different zones:

- PST: *Pacific Standard Time*, *Pakistan Standard Time*
- EST: *Eastern Standard Time (USA)*, *Eastern Standard Time (Australia)* and *Eastern Brazil Standard Time*

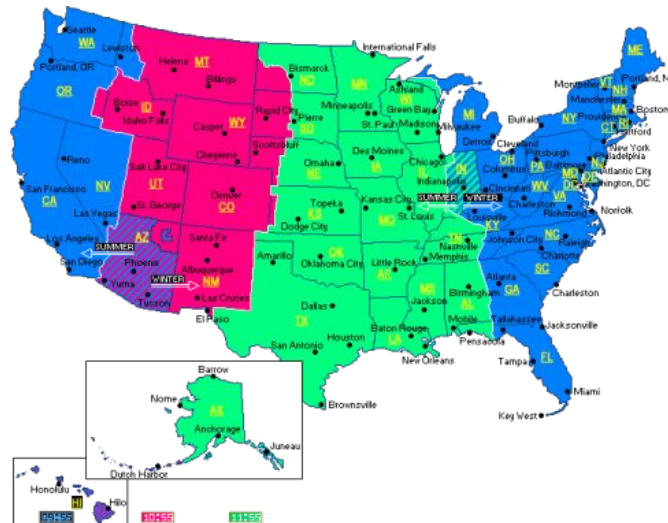
One zone can have multiple identifiers

- Central Europe Summer Time: CEST or CETDST

Names can be different on Operating Systems

# Problems

- Artificial time offset to save "daylight"
- Not all countries/areas use it
- Switches are not done at the same time for all areas
- There are plenty of exceptions



# Date/Time Functions in PHP 4 and PHP 5.0

- Uses Unix timestamp as base unit (seconds since 1970-01-01, 00:00 GMT)
- Only 32 bit integers for timestamps (1902 to 2038)
- Limited to only positive numbers on some Operating Systems (1970 to 2038)
- *strtotime()* is buggy and very complex
- No way of dealing correctly with timezones
- Some functions are Operating System dependent

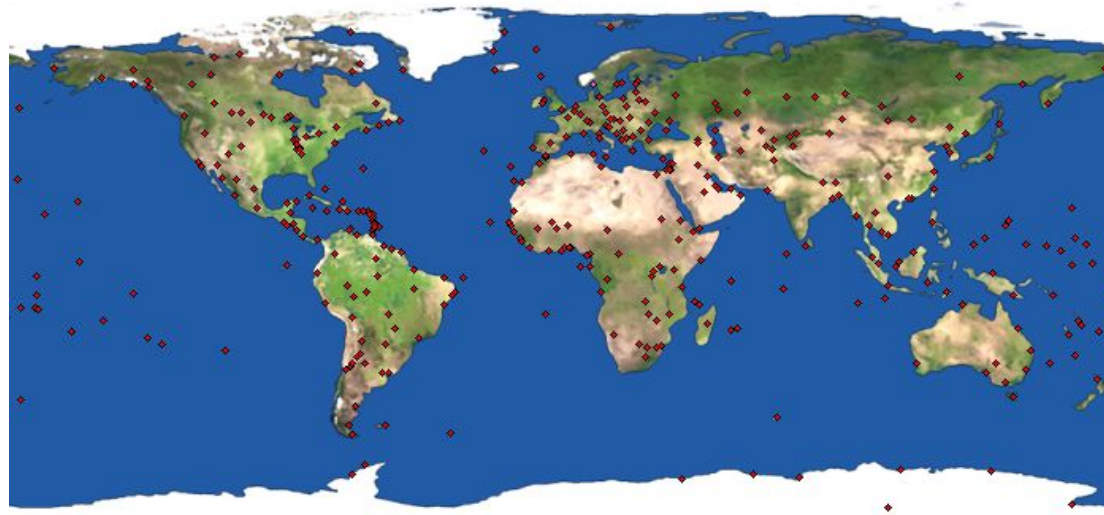


# Date/Time Functions in PHP 5.1

- 64 bit timestamps
- *strtotime()* has been rewritten
- Nothing is Operating System dependent
- Full support for timezones, DST, date modifications
- New format modifiers: e for timezone identifier and o for ISO Year
- Advanced date handling functions

# Date/Time Functions in PHP 5.1

- Bundled timezone database with 533 zones
- Not dependent on timezone abbreviations
- Timezones have the format: Continent/Location or Continent/Location/Sublocation - Like: Europe/Amsterdam, America/Indiana/Knox



# Parsing Dates - Take #2

Parsing strings for date time information with the *date\_create()* function:

```
<?php
    $ts = date_create("2005-07-11 22:16:50");
?>
```

This function will not return the timestamp as an integer, but instead returns a Date object which is a wrapper around a 64 bit integer (with some additional functionality ofcourse).

# Formatting Dates

## Formatting using format specifiers:

```
<?php
    date_default_timezone_set("Europe/Oslo");
    $ts = date_create("1979-12-31 09:15");
    echo date_format($ts, "D Y-m-d H:i:s - \I\S\O \W\Y: W/o"), "<br/>\n";
```

All format modifiers as supported by *date()* are supported too.

## Predefined formats:

```
<?php
    date_default_timezone_set("Europe/Oslo");
    $ts = date_create("December 22nd, 2005 15:41");
    echo date_format($ts, DATE_ISO8601), "<br/>\n";
    echo date_format($ts, DATE_RFC1036), "<br/>\n";
    echo date_format($ts, DATE_RSS), "<br/>\n";
```

# Modifying Dates

## Modifying dates and times:

```
<?php
    date_default_timezone_set("Europe/Oslo");
    $ts = date_create("now");
    echo $ts->format(DATE_RFC2822), "<br/>\n";

    echo $ts->modify("+2 days");
    echo $ts->format(DATE_RFC2822), "<br/>\n";

    echo $ts->modify("third month");
    echo $ts->format(DATE_RFC2822), "<br/>\n";

    echo $ts->modify("Friday +3 weeks");
    echo $ts->format(DATE_RFC2822), "<br/>\n";

    echo $ts->modify("next friday");
    echo $ts->format(DATE_RFC2822), "<br/>\n";
?>
```

# Using Timezones

Specifying timezone abbreviation while parsing:

```
<?php
    $ts = date_create("1978-12-22 09:15 CET");
?>
```

Using timezone abbreviations is deprecated, one should always use either a default timezone, or the full identifier.

Specifying timezone identifier while parsing:

```
<?php
    $ts = date_create("1978-12-22 09:15 Europe/Oslo");
?>
```

# Default Timezones

## Setting a default timezone:

```
<?php
    date_default_timezone_set("Europe/Oslo");
    $ts = date_create("1978-12-22 09:15");
    echo date_format($ts, "e");
?>
```

## Getting a default timezone:

```
<?php
    $default_identifier = date_default_timezone_get();
    echo $default_identifier;
?>
```

## Default timezone is 'guessed' in the following order:

- *date\_default\_timezone\_set()* value
- TZ environment variable
- php.ini's date.timezone setting
- System's rendering of timezone abbreviation

# Using Timezones

Creating a timezone resource:

```
<?php
    $tz = timezone_open("Asia/Singapore");
?>
```

Using the timezone when parsing a string with a date representation:

```
<?php
    $tz = timezone_open("Pacific/Honolulu");
    $ts = date_create("1978-12-22 09:15", $tz);
?>
```

A passed timezone object does not override a *parsed* timezone:

```
<?php
    $tz = timezone_open("Pacific/Honolulu");
    $ts1 = date_create("1978-12-22 09:15 CET", $tz);
    $ts2 = date_create("1978-12-22 09:15 Europe/Amsterdam", $tz);
?>
```



# Using Timezones

## Getting a timezone's name:

```
<?php
    $tz = timezone_open("Asia/Singapore");
    echo timezone_name_get($tz), ', ';

    $tz = timezone_open("CEST");
    echo $tz->getName();
?>
```

## Getting the current offset to GMT with a timezone for a specific date:

```
<?php
    $tz = timezone_open("Europe/Amsterdam");
    $d = date_create("2005-01-22 09:15");
    echo timezone_offset_get($tz, $d), ', ';
    $d->modify("+6 months");
    echo $tz->getOffset($d);
?>
```

# Using Timezones

Using the timezone when parsing a string with a date representation:

```
<?php
    $tz1 = timezone_open("Pacific/Honolulu");
    $tz2 = timezone_open("Europe/Amsterdam");
    $tz3 = timezone_open("Australia/Melbourne");

    $ts = date_create("1978-12-22 09:15", $tz1);
    echo $ts->getTimezone()->getName(), ': ',
        $ts->format(DATE_RFC822), "<br/>";

    $ts->setTimezone($tz2);
    echo $ts->getTimezone()->getName(), ': ',
        $ts->format(DATE_RFC822), "<br/>";

    date_timezone_set($ts, $tz3);
    echo timezone_name_get(date_timezone_get($ts)), ': ',
        date_format($ts, DATE_RFC822);
?>
```

# Timezones Utilities

## Creating a timezone resource:

```
<?php
    $tz = timezone_open("Europe/Amsterdam");
    $trs = timezone_transitions_get($tz);
    $trs = $tz->getTransitions();

    echo "<pre>\n";
    foreach ($trs as $tr) {
        printf("%20s %7d %d %s\n",
            $tr['time'], $tr['offset'],
            $tr['isdst'], $tr['abbr']);
    }
?>
```

# Timezones Utilities

## All supported timezone identifiers:

```
<?php
    $ids = timezone_identifiers_list();
    echo "Number of identifiers: ", count($ids), "<br/>";
    echo implode(", ", array_slice($ids, 0, 5)), '...';
    echo implode(", ", array_slice($ids, -5));
?>
```

## All supported timezone identifiers:

```
<?php
    $abbrs = timezone_abbreviations_list();
    echo "<pre>\n";
    foreach ($abbrs as $abbr => $id) {
        printf("%-4s %6d %d %s\n", strtoupper($abbr),
            $id['offset'], $id['dst'], $id['timezone_id']);
    }
?>
```