Zend_Locale Proposal - Thomas Weidner

<ac:macro ac:name="unmigrated-inline-wiki-markup"><ac:plain-text-body><![CDATA[

<ac:macro ac:name="unmigrated-inline-wiki-markup"><ac:plain-text-body><![CDATA[

Zend Framework: Zend_Locale Component Proposal

Proposed Component Name	Zend_Locale
Developer Notes	http://framework.zend.com/wiki/display/ZFDEV/Zend_Locale
Proposers	• Thomas Weidner • Gavin (Zend-liaison)
Revision	4.0 - 19 Sept 2006: Extracted translation classes to Zend_Translate 3.0 - 28 July 2006: Changed Class Skeleton and added new Use cases based on comments from Zend 2.1 - 11 July 2006: Changed Class Skeletons for matching actual work 2.0 - 20 Juni 2006: Reworked based on new Zend_Measure and Zend_Date Proposal. (wiki revision: 40)

Table of Contents

1. Overview
2. References
Locale Description Standard
LDML - Locale Data Markup Language
International Standards
ISO 639
ISO 3166
RFC 3066
I18N General
Mailing Lists
Unicode Discussion for upcoming PHP6
Discussions for Zend Framework
3. Component Requirements, Constraints, and Acceptance Criteria
4. Dependencies on Other Framework Components
5. Theory of Operation
Basics
Locale description format
Automatic locale recognition
Formatting with locales
Exception handling
Basically we have to implement/know the following standards
Outsourced or delayed functionality
Framework Components which use Zend_Locale
6. Milestones / Tasks
7. Class Index
8. Use Cases
Formatting
9. Class Skeletons

1. Overview

Zend_Locale is a basic wrapper for all I18N and L10N issues for the Zend Framework. It provides the framework with all locale related informations. All classes which should be locale-aware should implement Zend_Locale.

2. References

Locale Description Standard

Details to Locale Description Standard which has to be implemented

LDML - Locale Data Markup Language

LDML the Locale Data Markup Language is part of the CLDR Project and it describes the locales XLM based.

- Description of the LDML format
- The Common Language Data Repository Project
- Example XML for LDML - only download

International Standards

The following international standards must be used

ISO 639

International Language Code Definition
ISO 639-1 for 2 letter, ISO 639-2 for 3 letter language codes

- Home of ISO 639
- Language codes from ISO 639-1 and ISO 639-2

ISO 3166

International Country Code Definitions
ISO 3166-1 for 2 letter country codes

- Home of ISO 3166
- Country List from ISO 3166-1

RFC 3066

Identification of languages

- Memo of language identification
- Language Tag List for RFC 3066

I18N General

- I18N Gurus - Open international ressources directory

Mailing Lists

Mailinglist discussions in past

Unicode Discussion for upcoming PHP6

- PHP 6 Unicode Discussion

Discussions for Zend Framework

- Johannes Orth - first thoughts about locale support
- Jason Mynard - general I18N thoughts
- Garvin Vess - first summary of needed functions
- Thomas Weidner - summarize functionality for Zend_Locale
- Garvin Vess - simplifying I18N needs
- Garvin Vess - First response of pre-proposal for Zend_Locale

3. Component Requirements, Constraints, and Acceptance Criteria

- Wrapper functionality
- Lightweight and fast implementation
- Simple use
- Automatic recognition of language the browser requests

4. Dependencies on Other Framework Components

- Zend_Exception
- Zend_Cache - (optional)

5. Theory of Operation

Basics

Zend_Locale is a wrapper for all locale related informations, especially for the CLDR, in the Zend Framework. It has to be simple to use and as lightweight as possible.

Locale description format

Zend_Locale has to know all language codes (see ISO 639 and ISO 3166). Therefor a locale description format has to be implemented. As format the free avaiable LDML format will be used (part of CLDR), as it is already used by many Open Source Projects.

Automatic locale recognition

Zend_Locale has to recognize which language the browser is requesting (see ISO 639 and ISO 3166), and automatically find the best matching locale for the client. (Internal Fallback mechanism). Alternative it could check for the system locale (environment variables) when f.e. used at command line.

Formatting with locales

Zend_Locale_Format can tell other functions how to format datas in different locales. For this, the Module knows how different locales are formatting data. Data can be everything which is defined in LDML. This implements especially:

- Date / Time
- Currency
- Measurement
- Numbers
- and all other descriptions which are handled through the LDML

Exception handling

Zend_Locale_Exception handles and throws all exceptions which our Zend_Locale Classes or their subclasses will throw

Basically we have to implement/know the following standards

ISO 639 - Language Codes
ISO 3166 - Region Codes
RFC 3066 - Country Codes

Outsourced or delayed functionality

Zend_Locale_Collate defines how alphabetic sorting has to be done in different locales, for example german, greek or russian where you have more than 26 letters or the pronouncing is different. This Class will be delayed for later implementation

Framework Components which use Zend_Locale

- Zend_Date
- Zend_Calendar
- Zend_Measure
- Zend_Currency
- Zend_Http_Client
- Zend_Http_Server
- Zend_Format_Input

6. Milestones / Tasks

zone: Missing {zone-data:milestones}

7. Class Index

- Zend_Locale - base class
- Zend_Locale_Exception- exception handling

- Zend_Locale_Format - A standard interface for locale formatting
- Zend_Locale_Data - internal LDML handling class

8. Use Cases

Get default locale

```
$locale = new Zend_Locale();
print $locale->toString(); // Prints: de_AT
```

Set other language

```
$locale = new Zend_Locale();
print $locale->toString(); // Prints: de_AT
$locale->setLocale(Zend_Locale::fr_FR);
print $locale->toString(); // Prints: fr_FR
```

Get region and language

```
$locale = new Zend_Locale();
print $locale->getRegion(); // Prints: AT
print $locale->getLanguage(); // Prints: de
```

Get the accepted languages from a users browser

```
$locales = new Zend_Locale();
print_r($locales->getBrowser());
// returns f.e.
// array('de_DE' => 0.7, 'en_UK' => 0.4)
```

Formatting

Formatting numbers

```
$format = Zend_Locale_Format::getFormat('decimal');
// Could return '#,##0.00#'... would recognize f.e. '-123,456,789.12'

$input = '-123.456,78'; // german input string, german is actual set locale
$normalized = Zend_Locale_Format::getNumber($input);
// Would return -123456.78
```

Example useage of own date-format (Zend_Date will use Zend_Locale_Format internally)

```
$format = 'YYYY DD MM, EEEE';
$input = '12.06.2006';  // Standard locale german
$date = new Zend_Date($input);
$output = Zend_Date->toString($format);
// This would return '2006 12 06, vor Christus'
```

Example useage of a own defined format

```
    $input = 'AC, 12.06.2006 00:00:00.000'; // Normalized input
    $format = 'YYYY DD MM, EEEE';
    $output = Zend_Locale_Format->toFormat($input, $format);
    // returns '2006 12 06, vor Christus'
    $output = Zend_Locale_Format->toFormat($input, 'EE, YY.MM.DD');
    // returns 'NC, 12.06.06'
```

Checking if a locale string is a float value

```
     $input = '-123,456.78 e-18';  // Standard locale is de
     $boolean = Zend_Locale_Format::isFloat($input);
     // returns true
```

Getting a locale translated list of all countries

```
    $list = Zend_Locale_Format::getList('region'); // Standard locale is de
    // returns array ('UK' => 'Großbrittanien', 'US' => 'USA', 'FR' =>
'Frankreich',...)
```

9. Class Skeletons

```
class Zend_Locale_Exception extends Zend_Exception {}


class Zend_Locale {

    /**
     * Constructor for the Zend_Locale Module
     * Sets the default locale automaticaly with HTTP_ACCEPT_LANGUAGE
     * Can get the default locale source from Config or Connection if set
     *
     * @param object $connection - OPTIONAL a connection can be defined
     * @param string $locale - OPTIONAL default locale
     */
    public function __construct($connection,$locale) {}

    /**
     * Serializes the object
     */
    public function serialize() {}

    /**
     * Unserialized the object
     */
    public function unserialize($serialized) {}

    /**
     * Sets a new locale which has to be used for translation
     *
```

```
     * @param string $locale
     */
    public function setLocale($locale) {}

    /**
     * Returns the actual set locale for translation
     *
     * @return string $locale
     */
    public function getLocale() {}

    /**
     * Returns a list of all avaiable locales which can be used for translation
     * by this Framework in a well formed array. This depends on the locales
     * which are integrated in the source files.
     *
     * This function could, for example, be used to find the avaiable
     * locales, and then display only the supported country flags on a page.
     *
     * @return array $locales
     */
    public function getList() {}

    /**
     * Returns if a specified locale is avaiable for translation
     *
     * @param string $locale  - Locale to search
     * @return boolean
     */
    public function isAvaiable($locale) {}

    /**
     * Returns the default Language
     *
     * @return string $language
     */
    public function getLanguage() {}

    /**
     * Returns the default Regioncode
     *
     * @return string $regionCode
     */
    public function getRegion() {}

    /**
     * Returns an array with the locale datas the
     * users browser is accepting
     * Looks at HTTP_ACCEPT_LANGUAGE for default
     * Returns the encoded array of locales sorted by acceptance
     *
     * @param string $httpAcceptLanguage - optional
     * @param integer $range - optional
     *
     * @return array $acceptedLocales
     */
    public function getBrowser($httpAcceptLanguage, $range) {}

    /**
```

```
     * Returns an array with the locale datas the
     * environment is accepting
     * Returns the encoded array of locales sorted by acceptance
     *
     * @return array $acceptedLocales
     */
    public function getEnvironment() {}

    /**
     * Returns the default locale.
     * With type the source looking for the default locale could be seperated.
     *
     * Returns the default locale
     *
     * @param type $type - OPTIONAL (Browser, Environment)
     *
     * @return locale
     */
    public function getDefault($type) {}

    /**
     * _ is a shortcut for translate()
     * I know that Zend does not permitt Underscores in function names
     * but it is a standard shortcut in php for the translating function
     * It's Zend's decision if we could use this or not
     *
     * @see translate($string,$locale)
     */
    public function _($string,$locale) {}
}


class Zend_Locale_Format {

    /**
     * Returns the standard format for a specified or the default locale
     *
     * @param type $type     - Type of format (Decimal, Scientific, Percent, Date,
Time,...)
     * @param string $locale - Optional
     * @return mixed
     */
    public static function getFormat($type, $locale) {}

    /**
     * Formats an input, returning the formatted string/value
     * But needs an normalized input !!
     *
     * @param format $format     - Format definition string
     * @param string $locale - Optional
     * @return mixed
     */
    public static function toFormat($format, $locale) {}

    /**
     * Returns an integer number representation from a string locale-aware
     *
     * @param string $input   Inputstring to parse
     * @param string $locale  OPTIONAL locale for parsing
```

```
     * @return integer
     */
    public static function getNumber($input, $locale) {}

    /**
     * Returns true if a string includes an locale-aware number
     *
     * @param string $input   Inputstring to parse
     * @param string $locale  OPTIONAL locale for parsing
     * @return boolean
     */
    public static function isNumber($input, $locale) {}

    /**
     * Returns an date string representation from a string locale-aware
     *
     * @param string $input   Inputstring to parse
     * @param string $locale  OPTIONAL locale for parsing
     * @return integer
     */
    public static function getDate($input, $locale) {}

    /**
     * Returns true if string includes a locale-aware date
     *
     * @param string $input   Inputstring to parse
     * @param string $locale  OPTIONAL locale for parsing
     * @return integer
     */
    public static function isDate($input, $locale) {}

    /**
     * Returns an time string representation from a string locale-aware
     *
     * @param string $input   Inputstring to parse
     * @param string $locale  OPTIONAL locale for parsing
     * @return integer
     */
    public static function getTime($input, $locale) {}

    /**
     * Returns true if string includes a locale-aware time
     *
     * @param string $input   Inputstring to parse
     * @param string $locale  OPTIONAL locale for parsing
     * @return integer
     */
    public static function isTime($input, $locale) {}

    /**
     * Returns an calendar string representation from a string locale-aware
     * (Date, Time & Era)
     *
     * @param string $input   Inputstring to parse
     * @param string $locale  OPTIONAL locale for parsing
     * @return integer
     */
    public static function getCalendar($input, $locale) {}
```

```
    /**
     * Returns true if string includes a locale-aware calendar
     *
     * @param string $input   Inputstring to parse
     * @param string $locale  OPTIONAL locale for parsing
     * @return integer
     */
    public static function isCalendar($input, $locale) {}

    /**
     * Returns the float number representation from a string locale-aware
     *
     * @param string $input   Inputstring to parse
     * @param string $locale  OPTIONAL locale for parsing
     * @return float
     */
    public static function getFloat($input, $locale) {}

    /**
     * Returns true if string includes a locale-aware floating number
     *
     * @param string $input   Inputstring to parse
     * @param string $locale  OPTIONAL locale for parsing
     * @return integer
     */
    public static function isFloat($input, $locale) {}

    /**
     * Returns a description or definition list from LDML
     *
     * @param type $type     - Optional type of list (Language, Script, Region,
Currency,...)
     * @param string $locale - Optional
     * @return array
     */
    public static function getList($type, $locale) {}

    /**
     * Returns a single description or definition from LDML
     *
     * @param type $type     - Optional type of list (Language, Script, Region,
Currency,...)
     * @param string $locale - Optional
     * @return mixed
     */
    public static function getValue($type, $locale) {}

    /**
     * Throws a exeption, faster than preloading
     */
    public static function throwExceptoin ($message) {}
}



class Zend_Locale_Data {

    /**
     * locale files
```

```
     *
     * @var ressource
     * @access private
     */
    private static $_ldml = array();


    /**
     * list of values which are collected
     *
     * @var array
     * @access private
     */
    private static $_list = array();


    /**
     * Read the content from locale
     *
     * @param  string $locale
     * @param  string $path
     * @param  string $attribute
     * @param  string $value
     * @access private
     */
    private static function _readFile($locale, $path, $attribute, $value) {}

    /**
     * Find possible routing to other path or locale
     *
     * @param  string $locale
     * @param  string $path
     * @param  string $attribute
     * @param  string $value
     * @access private
     */
    private static function _findRoute($locale, $path, $attribute, $value) {}


    /**
     * Read the right LDML file
     *
     * @param  string $locale
     * @param  string $path
     * @param  string $attribute
     * @param  string $value
     * @access private
     */
    private static function _getFile($locale, $path, $attribute, $value) {}


    /**
     * Read the LDML, get a single path defined value
     * Possible Values for path:
     * 'language', 'script', 'territory', 'variant', 'key', 'type', 'characters',
'layout',
     * and much more... take a look at the documentation
     *
     * @param  string $locale
```

```
     * @param  string $path
     * @param  string $attribute
     * @param  string $value
     * @return array of string
     * @access public
     */
    public static function getContent($locale, $path, $value) {}

    /**
     * Throws a exeption, faster than preloading
     */
    public static function throwExceptoin ($message) {}
```

```
}
```

]]></ac:plain-text-body></ac:macro>
]]></ac:plain-text-body></ac:macro>