Zend_Date 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_Date Component Proposal

Proposed Component Name	Zend_Date
Developer Notes	http://framework.zend.com/wiki/display/ZFDEV/Zend_Date
Proposers	Thomas Weidner Gavin (Zend-liaison)
Revision	2.0 - 21 July 2006: Reworked proposal (changed almost everything) 1.0 - 11 Juni 2006: Initial proposal (wiki revision: 23)

Table of Contents

1. Overview

Zend_Date is the basic library for the handling of all date, time and calendar-related issues.

2. References

Mailing List

- Alexandre Riveira - First Zend_Date announcement, incomplete

Date and Time related links

- Date and Time Formats from W3C
- ISO 8601
- Date and time notation by country
- Date and Time in Internet - Timestamps
- NPL - Time and Frequency Labratory

- Sources for Time Zone and Daylight Saving Time Data
- Critical and Significant dates
- Program for Atomic Scnyronize

Java

- Java Date Class API
- Java Calendar Class API

Python

- Python Implementation of DateTime
- Python Implementation of Date
- Python Implementation of Time

Ruby

- Ruby implementation of DateTime
- Ruby implementation of Date
- Ruby implementation of Time

Calendars

- Julian and gregorian day numbers
- Overview of different calendar formats

3. Component Requirements, Constraints, and Acceptance Criteria

- Wrapper functionality
- Lightweight and fast implementation
- Conversion between different locales for date / time
- Conversion between different calendar-formats
- Formatting for date, time, calendar using locales

4. Dependencies on Other Framework Components

- Zend_Exception
- Zend_Locale_Format

5. Theory of Operation

Basics

Zend_Date is a basic wrapper for all date, time functions. It implements also calendar and some other additional functionality.

Locale aware

Zend_Date formats the output with locales. It can output the standard format for different uses f.e. ISO date, unix timestamp, SQL dateformat, ...
Zend_Date also can convert between different date / time formats and locales

Calendar functions

Zend_Date can convert between and display in different calendar formats. Julian, Gregorian and so on...

Computed functions

Dates can be added or substracted from other dates. Also a difference between dates can be computed

Standards

Zend_Date has to be aware of the following standards.

- ISO 8601 - Date Time Definition
- RFC 3339 - Internet Date/Time
- RFC 822 - Arpa Mail Format (Date/Time)

6. Milestones / Tasks

zone: Missing {zone-data:milestones}

7. Class Index

- Zend_Date_Exception
- Zend_Date

8. Use Cases

Define actual Date

```
$dateobj = Zend_Date(); // Without input the actual date is used
```

Define Date where Input is an ISO Date

```
$date = '2006-05-23 10:05:23.000';  // Input date in ISO Format
$dateobj = Zend_Date($date);
```

Define Date with Unix Timestamp

```
$date = '143465234';    // Input date in Unix Timestamp Format
$dateobj = Zend_Date($date,Zend_Date::UNIX);
```

Define Date where Input is an Locale Defined Datestring

```
    $date = '23.05.2006 10:05:23.000';  // Input date in Locale Format
    $dateobj = Zend_Date($date,Zend_Locale::DE_DE);
```

Define Date where Input is an Arab Date

```
    $date = '23.05.2006 10:05:23.000';  // Input date in Locale Format
    $dateobj = Zend_Date($date,Zend_Date::CALENDAR_ARABIAN);
```

Convert Date to Unix Timestamp

```
    $date = '2006-05-23 10:05:23.000';  // Input date in ISO Format
    $dateobj = Zend_Date($date);
    echo $dateobj->getTimeStamp();       // Outputs Unix Timestamp '143465234'
```

Convert Date to ISO Format

```
    $date = '143465234';                // Input date in Unix Timestamp Format
    $dateobj = Zend_Date($date,Zend_Date::UNIX);
    echo $dateobj->getIso();             // Outputs Standard ISO Format '2006-05-23
10:05:23'
```

Get actual monthname

```
    $dateobj = Zend_Date();      // Actual date
    echo $dateobj->getMonth('Zend_Date::MONTHNAME');  // Outputs the full name of the
month with actual locale outputs f.e. 'May'
```

Print date formatted with actual locale

```
    $dateobj = Zend_Date();  // Actual date
    echo $dateobj->getMonth('Zend_Date::MONTHNAME',Zend_Date::DE_DE);   // Outputs the
full name of the month with other locale outputs f.e. 'Mai'
```

Print date formatted with specified locale

```
    $dateobj = Zend_Date();  // Actual date
    echo $dateobj->getDate(Zend_Locale::DE_DE);  // Outputs the date formatted with
given locale outputs f.e. '23.05.2006'
```

9. Class Skeletons

```
class Zend_Date_Exception extends Zend_Exception {}
```

```
class Zend_Date implements Serializable {

    /**
     * Generates the standard date object
     * could be
     *    - Unix timestamp
     *    - ISO
     *    - Locale
     *
     * @param $date string - OPTIONAL date string depending on $parameter
     * @param $locale string   - OPTIONAL locale for parsing input
     * @param $parameter mixed - OPTIONAL defines the input format of $date
     * @return object
     */
    public function __construct($date, $locale, $parameter) {}

    /**
     * Serialization Interface
     */
    public function serialize() {}
    public function unserialize() {}

    /**
     * Returns a string representation of the object
     *
     * @param $locale string   - OPTIONAL locale for parsing input
     * @param $format - OPTIONAL an rule for formatting the output
     * @return string
     */
    public function toString($locale, $format);
    public function __toString($locale, $format);

    /**
     * Returns a integer representation of the object
     *
     * @return integer
     */
    public function toValue();

    /**
     * Adds a date to another date. Could add f.e. minutes, hours, days, months to a
date object
     *
     * @param $date object     - date which shall be added to our actual date object
     * @param $part datepart   - OPTIONAL datepart to add
     * @return object
     */
    public function add($date, $part) {}

    /**
     * Substracts a date from another date. Could sub f.e. minutes, hours, days from a
date object
     *
     * @param $date object     - date which shall be substracted to our actual date
object
     * @param $part datepart   - OPTIONAL datepart to substract
     * @return object
```

```
     */
    public function sub($date, $part) {}

    /**
     * Compares a date with another date. Returns a date object with the difference
date
     *
     * @param $date object   - date which shall be compared with our actual date
object
     * @param $part datepart - OPTIONAL datepart to set
     * @return object
     */
    public function compare($date, $part) {}

    /**
     * Returns a duplicate date object
     *
     * @param $part datepart - OPTIONAL the part of date to clone
     * @return object
     */
    public function clone($part) {}

    /**
     * Sets the given date as new date
     *
     * @param $date string   - date which shall be our new date object
     * @param $part datepart - OPTIONAL datepart to set
     * @return object
     */
    public function set($date, $part) {}

    /**
     * Returns true when both date objects have equal dates set
     *
     * @param $date object
     * @return boolean
     */
    public function equals($date) {}

    /**
     * Returns the maximum date or datepart for the set date object/calendar type
     *
     * @param $part  datepart / string - OPTIONAL
     * @return object / datepart
     */
    public function getMaximum($part) {}

    /**
     * Returns the minimum date or datepart for the set date object/calendar type
     *
     * @param $part  datepart / string - OPTIONAL
     * @return object / datepart
     */
    public function getMinimum($part) {}

    /**
     * Returns the first day of the week
     *
     * @param $locale string   - OPTIONAL locale for parsing input
```

```
     * @param $format string   - OPTIONAL how to return the day (Number, Name,
DateObject)
     * @return mixed - object / datepart / integer
     */
    public function getFirstDayOfWeek($locale, $format) {}

    /**
     * Returns the time
     * Alias for clone(Zend_Date::TIME);
     *
     * @return object
     */
    public function getTime();

    /**
     * Sets a new time
     * Alias for set($time, Zend_Date::TIME);
     *
     * @param $time string     - OPTIONAL time to set, when null the actual time is
set
     * @param $locale string   - OPTIONAL locale for parsing input
     * @param $format          - OPTIONAL an rule for parsing the input
     * @return object
     */
    public function setTime($time, $locale, $format);

    /**
     * Adds a time
     * Alias for add($time,Zend_Date::TIME);
     *
     * @param $time object     - OPTIONAL time to add, when null the actual time is
add
     * @return object
     */
    public function addTime($time);

    /**
     * Substracts a time
     * Alias for sub($time,Zend_Date::TIME);
     *
     * @param $time object     - OPTIONAL time to sub, when null the actual time is
sub
     * @return object
     */
    public function subTime($time);

    /**
     * Compares only the time, returning the difference
     * Alias for compare($time,Zend_Date::TIME);
     *
     * @param $time object     - OPTIONAL time to compare, when null the actual time
is used for compare
     * @return object
     */
    public function compareTime($time);

    /**
     * Returns the date
     * Alias for clone(Zend_Date::DATE);
```

```
     *
     * @return object
     */
    public function getDate();

    /**
     * Sets a new date
     * Alias for set($date, Zend_Date::DATE);
     *
     * @param $date string     - OPTIONAL date to set, when null the actual date is
set
     * @param $locale string   - OPTIONAL locale for parsing input
     * @param $format          - OPTIONAL an rule for parsing the input
     * @return object
     */
    public function setDate($date, $locale, $format);

    /**
     * Adds a date
     * Alias for add($date,Zend_Date::DATE);
     *
     * @param $date object     - OPTIONAL date to add, when null the actual date is
add
     * @return object
     */
    public function addDate($time);

    /**
     * Substracts a date
     * Alias for sub($date,Zend_Date::DATE);
     *
     * @param $date object     - OPTIONAL date to sub, when null the actual date is
sub
     * @return object
     */
    public function subDate($date);

    /**
     * Compares only the date, returning the difference date
     * Alias for compare($date,Zend_Date::DATE);
     *
     * @param $date object     - OPTIONAL date to compare, when null the actual date
is used for compare
     * @return object
     */
    public function compareDate($date);

    /**
     * Returns the unix timestamp
     *
     * @return timestamp
     */
    public function getTimeStamp();

    /**
     * Sets a new timestamp
     * Alias for set($stamp);
     *
     * @param $timestamp timestamp  - OPTIONAL timestamp to set, when null the actual
```

```
timestamp is set
     * @return object
     */
    public function setTimeStamp($timestamp);

    /**
     * Adds a timestamp
     * Alias for add($timestamp);
     *
     * @param $timestamp timestamp  - OPTIONAL timestamp to add, when null the actual
timestamp is add
     * @return object
     */
    public function addTimeStamp($timestamp);

    /**
     * Substracts a timestamp
     * Alias for sub($timestamp);
     *
     * @param $timestamp timestamp  - OPTIONAL timestamp to sub, when null the actual
timestamp is sub
     * @return object
     */
    public function subTimeStamp($timestamp);

    /**
     * Compares two timestamps, returning the difference timestamp
     * Alias for compare($timestamp);
     *
     * @param $timestamp timestamp - OPTIONAL timestamp to compare, when null the
actual timestamp is used for compare
     * @return object
     */
    public function compareTimeStamp($timestamp);

    /**
     * Returns a ISO8601 formatted date - ISO is locale-independent
     *
     * @param $format - OPTIONAL an rule for formatting the output for different ISO
Formats
     * @return string
     */
    public function getIso($format);

    /**
     * Sets a new ISOdate
     * Alias for set($date);
     *
     * @param $date string     - OPTIONAL ISOdate to set, when null the actual date is
set
     * @param $format          - OPTIONAL an rule for parsing the ISOinput
     * @return object
     */
    public function setIso($date, $format);

    /**
     * Adds a ISOdate
     * Alias for add($date);
     *
```

```
     * @param $date string  - OPTIONAL ISOdate to add, when null the actual date is
add
     * @param $format       - OPTIONAL an rule for parsing the ISOinput
     * @return object
     */
    public function addIso($date, $format);

    /**
     * Substracts a ISOdate
     * Alias for sub($date);
     *
     * @param $date string  - OPTIONAL ISOdate to sub, when null the actual date is
sub
     * @param $format       - OPTIONAL an rule for parsing the ISOinput
     * @return object
     */
    public function subIso($date, $format);

    /**
     * Compares IsoDate with date object, returning the difference date
     * Alias for compare($date);
     *
     * @param $date string - OPTIONAL ISOdate to compare, when null the actual date is
used for compare
     * @return object
     */
    public function compareIso($date);

    /**
     * Returns a RFC822 formatted date - RFC822 is locale-independent
     *
     * @return string
     */
    public function getArpa();

    /**
     * Sets a new RFC822 formatted date
     * Alias for set($date);
     *
     * @param $date string     - OPTIONAL RFC822 date to set, when null the actual
date is set
     * @return object
     */
    public function setArpa($date);

    /**
     * Adds a RFC822 formatted date
     * Alias for add($date);
     *
     * @param $date string  - OPTIONAL RFC822 date to add, when null the actual date
is add
     * @return object
     */
    public function addArpa($date);

    /**
     * Substracts a RFC822 formatted date
     * Alias for sub($date);
     *
```

```
     * @param $date string  - OPTIONAL RFC822 date to sub, when null the actual date
is sub
     * @return object
     */
    public function subArpa($date);

    /**
     * Compares a RFC822 formatted date with date object, returning the difference
date
     * Alias for compare($date);
     *
     * @param $date string - OPTIONAL RFC822 date to compare, when null the actual
date is used for compare
     * @return object
     */
    public function compareArpa($date);

    /**
     * Returns a UTC/GMT formatted date
     *
     * @param $timezone string   - OPTIONAL locale or timezone for setting output
timezone
     * @return string
     */
    public function getUtc($timezone);

    /**
     * Sets a new UTC/GMT formatted date
     * Alias for set($date);
     *
     * @param $date string     - OPTIONAL UTC/GMT formatted date to set, when null the
actual date is set
     * @return object
     */
    public function setUtc($date);

    /**
     * Adds a UTC/GMT formatted date
     * Alias for add($date);
     *
     * @param $date string  - OPTIONAL UTC/GMT formatted date to add, when null the
actual date is add
     * @return object
     */
    public function addUtc($date);

    /**
     * Substracts a UTC/GMT formatted date
     * Alias for sub($date);
     *
     * @param $date string  - OPTIONAL UTC/GMT formatted date to sub, when null the
actual date is sub
     * @return object
     */
    public function subUtc($date);

    /**
     * Compares UTC/GMT formatted date with date object, returning the difference date
     * Alias for compare($date);
```

```
     *
     * @param $date string - OPTIONAL UTC/GMT date to compare, when null the actual
date is used for compare
     * @return object
     */
    public function compareUtc($date);

    /**
     * Returns the time of sunrise for this locale or an optional location
     *
     * @param  $location array - OPTIONAL location of sunrise
     * @return object
     */
    public function getSunRise($location);

    /**
     * Returns the time of sunset for this locale or an optional location
     *
     * @param  $location array - OPTIONAL location of sunset
     * @return object
     */
    public function getSunSet($location);

    /**
     * Returns an array with all sun-infos for a time and location
     *
     * @param  $location array - location of suninfo
     * @return object
     */
    public function getSunInfo($location);

    /**
     * Returns the timezone
     *
     * @param $locale string   - OPTIONAL locale timezone
     * @return object
     */
    public function getTimeZone($locale);

    /**
     * Sets the timezone
     *
     * @param $timezone timezone - OPTIONAL timezone to set
     * @param $locale string   - OPTIONAL locale for timezone
     * @return object
     */
    public function setTimeZone($locale);

    /**
     * Returns if the date is a leap year
     *
     * @return boolean
     */
    public function isLeapYear();

    /**
     * Returns if the date is todays date
     *
     * @return boolean
```

```
     */
    public function isToday();

    /**
     * Returns if the date is yesterdays date
     *
     * @return boolean
     */
    public function isYesterday();

    /**
     * Returns if the date is tomorrows date
     *
     * @return boolean
     */
    public function isTomorrow();

    /**
     * Returns if the date is after this date
     *
     * @param $date object     - date to compare
     * @return boolean
     */
    public function isAfter($date);

    /**
     * Returns if the date is befor this date
     *
     * @param $date object     - date to compare
     * @return boolean
     */
    public function isBefor($date);

    /**
     * Returns actual date as object
     *
     * @return object
     */
    public function now();

    /**
     * Returns the year
     * Alias for clone(Zend_Date::YEAR);toString();
     *
     * @return string
     */
    public function getYear();

    /**
     * Sets a new year
     * Alias for set($year, Zend_Date::YEAR);
     *
     * @param $year string/integer - OPTIONAL year to set, when null the actual year
is set
     * @param $locale string   - OPTIONAL locale for parsing input
     * @return object
     */
    public function setYear($year, $locale);
```

```
    /**
     * Adds a year
     * Alias for add($year,Zend_Date::YEAR);
     *
     * @param $year object     - OPTIONAL year to add, when null the actual year is
add
     * @return object
     */
    public function addYear($year);

    /**
     * Substracts a year
     * Alias for sub($year,Zend_Date::YEAR);
     *
     * @param $year object     - OPTIONAL year to sub, when null the actual year is
sub
     * @return object
     */
    public function subYear($year);

    /**
     * Compares only the year part, returning the difference
     * Alias for compare($year,Zend_Date::YEAR);
     *
     * @param $year string/integer - OPTIONAL year to compare, when null the actual
year is used for compare
     * @return string
     */
    public function compareYear($year);

    /**
     * Compares only the year part, returning boolean true
     * Alias for compare($year,Zend_Date::YEAR);
     *
     * @param $year string/integer - OPTIONAL year to compare, when null the actual
year is used for compare
     * @return string
     */
    public function isYear($year);

    /**
     * Returns the month
     * Alias for clone(Zend_Date::MONTH);toString();
     *
     * @return string
     */
    public function getMonth();

    /**
     * Sets a new month
     * Alias for set($month, Zend_Date::MONTH);
     *
     * @param $month string/integer - OPTIONAL month to set, when null the actual
month is set
     * @param $locale string   - OPTIONAL locale for parsing input
     * @return object
     */
    public function setMonth($month, $locale);
```

```
    /**
     * Adds a month
     * Alias for add($month,Zend_Date::MONTH);
     *
     * @param $month object     - OPTIONAL month to add, when null the actual month is
add
     * @return object
     */
    public function addMonth($month);

    /**
     * Substracts a month
     * Alias for sub($month,Zend_Date::MONTH);
     *
     * @param $month object     - OPTIONAL month to sub, when null the actual month is
sub
     * @return object
     */
    public function subMonth($month);

    /**
     * Compares only the month part, returning the difference
     * Alias for compare($month,Zend_Date::MONTH);
     *
     * @param $month string/integer - OPTIONAL month to compare, when null the actual
month is used for compare
     * @return string
     */
    public function compareMonth($month);

    /**
     * Compares only the month part, returning boolean true
     * Alias for compare($month,Zend_Date::MONTH);
     *
     * @param $month string/integer - OPTIONAL month to compare, when null the actual
month is used for compare
     * @return string
     */
    public function isMonth($month);

    /**
     * Returns the day
     * Alias for clone(Zend_Date::DAY);toString();
     *
     * @return string
     */
    public function getDay();

    /**
     * Sets a new day
     * Alias for set($day, Zend_Date::DAY);
     *
     * @param $day string/integer - OPTIONAL day to set, when null the actual day is
set
     * @param $locale string   - OPTIONAL locale for parsing input
     * @return object
     */
    public function setDay($day, $locale);
```

```
    /**
     * Adds a day
     * Alias for add($day,Zend_Date::DAY);
     *
     * @param $day object     - OPTIONAL day to add, when null the actual day is add
     * @return object
     */
    public function addDay($day);

    /**
     * Substracts a day
     * Alias for sub($day,Zend_Date::DAY);
     *
     * @param $day object     - OPTIONAL day to sub, when null the actual day is sub
     * @return object
     */
    public function subDay($day);

    /**
     * Compares only the day part, returning the difference
     * Alias for compare($day,Zend_Date::DAY);
     *
     * @param $day string/integer - OPTIONAL day to compare, when null the actual day
is used for compare
     * @return string
     */
    public function compareDay($day);

    /**
     * Compares only the day part, returning boolean true
     * Alias for compare($day,Zend_Date::DAY);
     *
     * @param $day string/integer - OPTIONAL day to compare, when null the actual day
is used for compare
     * @return string
     */
    public function isDay($day);

    /**
     * Returns the hour
     * Alias for clone(Zend_Date::HOUR);toString();
     *
     * @return string
     */
    public function getHour();

    /**
     * Sets a new hour
     * Alias for set($hour, Zend_Date::HOUR);
     *
     * @param $hour string/integer - OPTIONAL hour to set, when null the actual hour
is set
     * @param $locale string   - OPTIONAL locale for parsing input
     * @return object
     */
    public function setHour($hour, $locale);

    /**
     * Adds a hour
```

```
     * Alias for add($hour,Zend_Date::HOUR);
     *
     * @param $hour object     - OPTIONAL hour to add, when null the actual hour is
add
     * @return object
     */
    public function addHour($hour);

    /**
     * Substracts a hour
     * Alias for sub($hour,Zend_Date::HOUR);
     *
     * @param $hour object     - OPTIONAL hour to sub, when null the actual hour is
sub
     * @return object
     */
    public function subHour($hour);

    /**
     * Compares only the hour part, returning the difference
     * Alias for compare($hour,Zend_Date::HOUR);
     *
     * @param $hour string/integer - OPTIONAL hour to compare, when null the actual
hour is used for compare
     * @return string
     */
    public function compareHour($hour);

    /**
     * Compares only the hour part, returning boolean true
     * Alias for compare($hour,Zend_Date::HOUR);
     *
     * @param $hour string/integer - OPTIONAL hour to compare, when null the actual
hour is used for compare
     * @return string
     */
    public function isHour($hour);

    /**
     * Returns the minute
     * Alias for clone(Zend_Date::MINUTE);toString();
     *
     * @return string
     */
    public function getMinute();

    /**
     * Sets a new minute
     * Alias for set($minute, Zend_Date::MINUTE);
     *
     * @param $minute string/integer - OPTIONAL minute to set, when null the actual
minute is set
     * @param $locale string   - OPTIONAL locale for parsing input
     * @return object
     */
    public function setMinute($minute, $locale);

    /**
     * Adds a minute
```

```
     * Alias for add($minute,Zend_Date::MINUTE);
     *
     * @param $minute object     - OPTIONAL minute to add, when null the actual minute
is add
     * @return object
     */
    public function addMinute($minute);

    /**
     * Substracts a minute
     * Alias for sub($minute,Zend_Date::MINUTE);
     *
     * @param $minute object     - OPTIONAL minute to sub, when null the actual minute
is sub
     * @return object
     */
    public function subMinute($minute);

    /**
     * Compares only the minute part, returning the difference
     * Alias for compare($minute,Zend_Date::MINUTE);
     *
     * @param $minute string/integer - OPTIONAL minute to compare, when null the
actual minute is used for compare
     * @return string
     */
    public function compareMinute($minute);

    /**
     * Compares only the minute part, returning boolean true
     * Alias for compare($minute,Zend_Date::MINUTE);
     *
     * @param $minute string/integer - OPTIONAL minute to compare, when null the
actual minute is used for compare
     * @return string
     */
    public function isMinute($minute);

    /**
     * Returns the second
     * Alias for clone(Zend_Date::SECOND);toString();
     *
     * @return string
     */
    public function getSecond();

    /**
     * Sets a new second
     * Alias for set($second, Zend_Date::SECOND);
     *
     * @param $second string/integer - OPTIONAL second to set, when null the actual
second is set
     * @param $locale string   - OPTIONAL locale for parsing input
     * @return object
     */
    public function setSecond($second, $locale);

    /**
     * Adds a second
```

```
     * Alias for add($second,Zend_Date::SECOND);
     *
     * @param $second object     - OPTIONAL second to add, when null the actual second
is add
     * @return object
     */
    public function addSecond($second);

    /**
     * Substracts a second
     * Alias for sub($second,Zend_Date::SECOND);
     *
     * @param $second object     - OPTIONAL second to sub, when null the actual second
is sub
     * @return object
     */
    public function subSecond($second);

    /**
     * Compares only the second part, returning the difference
     * Alias for compare($second,Zend_Date::SECOND);
     *
     * @param $second string/integer - OPTIONAL second to compare, when null the
actual second is used for compare
     * @return string
     */
    public function compareSecond($second);

    /**
     * Compares only the second part, returning boolean true
     * Alias for compare($second,Zend_Date::SECOND);
     *
     * @param $second string/integer - OPTIONAL second to compare, when null the
actual second is used for compare
     * @return string
     */
    public function isSecond($second);

    /**
     * Returns the millisecond
     * Alias for clone(Zend_Date::MILLISECOND);toString();
     *
     * @return string
     */
    public function getMilliSecond();

    /**
     * Sets a new millisecond
     * Alias for set($millisecond, Zend_Date::MILLISECOND);
     *
     * @param $millisecond string/integer - OPTIONAL millisecond to set, when null the
actual millisecond is set
     * @param $locale string   - OPTIONAL locale for parsing input
     * @return object
     */
    public function setMilliSecond($millisecond, $locale);

    /**
     * Adds a millisecond
```

```
     * Alias for add($millisecond,Zend_Date::MILLISECOND);
     *
     * @param $millisecond object     - OPTIONAL millisecond to add, when null the
actual millisecond is add
     * @return object
     */
    public function addMilliSecond($millisecond);

    /**
     * Substracts a millisecond
     * Alias for sub($millisecond,Zend_Date::MILLISECOND);
     *
     * @param $millisecond object     - OPTIONAL millisecond to sub, when null the
actual millisecond is sub
     * @return object
     */
    public function subMilliSecond($millisecond);

    /**
     * Compares only the millisecond part, returning the difference
     * Alias for compare($millisecond,Zend_Date::MILLISECOND);
     *
     * @param $millisecond string/integer - OPTIONAL millisecond to compare, when null
the actual millisecond is used for compare
     * @return string
     */
    public function compareMilliSecond($millisecond);

    /**
     * Compares only the millisecond part, returning boolean true
     * Alias for compare($millisecond,Zend_Date::MILLISECOND);
     *
     * @param $millisecond string/integer - OPTIONAL millisecond to compare, when null
the actual millisecond is used for compare
     * @return string
     */
    public function isMilliSecond($millisecond);

    /**
     * Returns the era
     * Alias for clone(Zend_Date::ERA);toString();
     *
     * @return string
     */
    public function getEra();

    /**
     * Sets a new era
     * Alias for set($era, Zend_Date::ERA);
     *
     * @param $era string/integer - OPTIONAL era to set, when null the actual era is
set
     * @param $locale string   - OPTIONAL locale for parsing input
     * @return object
     */
    public function setEra($era, $locale);

    /**
     * Adds a era
```

```
     * Alias for add($era,Zend_Date::ERA);
     *
     * @param $era object     - OPTIONAL era to add, when null the actual era is add
     * @return object
     */
    public function addEra($era);

    /**
     * Substracts a era
     * Alias for sub($era,Zend_Date::ERA);
     *
     * @param $era object     - OPTIONAL era to sub, when null the actual era is sub
     * @return object
     */
    public function subEra($era);

    /**
     * Compares only the era part, returning the difference
     * Alias for compare($era,Zend_Date::ERA);
     *
     * @param $era string/integer - OPTIONAL era to compare, when null the actual era
is used for compare
     * @return string
     */
    public function compareEra($era);

    /**
     * Compares only the era part, returning boolean true
     * Alias for compare($era,Zend_Date::ERA);
     *
     * @param $era string/integer - OPTIONAL era to compare, when null the actual era
is used for compare
     * @return string
     */
    public function isEra($era);

    /**
     * Returns the week
     * Alias for clone(Zend_Date::WEEK);toString();
     *
     * @return string
     */
    public function getWeek();

    /**
     * Sets a new week
     * Alias for set($week, Zend_Date::WEEK);
     *
     * @param $week string/integer - OPTIONAL week to set, when null the actual week
is set
     * @param $locale string   - OPTIONAL locale for parsing input
     * @return object
     */
    public function setWeek($week, $locale);

    /**
     * Adds a week
     * Alias for add($week,Zend_Date::WEEK);
     *
```

```
     * @param $week object     - OPTIONAL week to add, when null the actual week is
add
     * @return object
     */
    public function addWeek($week);

    /**
     * Substracts a week
     * Alias for sub($week,Zend_Date::WEEK);
     *
     * @param $week object     - OPTIONAL week to sub, when null the actual week is
sub
     * @return object
     */
    public function subWeek($week);

    /**
     * Compares only the week part, returning the difference
     * Alias for compare($week,Zend_Date::WEEK);
     *
     * @param $week string/integer - OPTIONAL week to compare, when null the actual
week is used for compare
     * @return string
     */
    public function compareWeek($week);

    /**
     * Compares only the week part, returning boolean true
     * Alias for compare($week,Zend_Date::WEEK);
     *
     * @param $week string/integer - OPTIONAL week to compare, when null the actual
week is used for compare
     * @return string
     */
    public function isWeek($week);

    /**
     * Returns the day of week of our Date Object
     *
     * @param $format string - OPTIONAL defines the format in which the day of the
week should be outputted (1/sunday/...)
     * @param $locale string - OPTIONAL defines the locale in which the day of the
week should be outputted
     * @return string
     */
    public function getDayOfWeek($locale, $format) {}

    /**
     * Returns the day of year of our Date Object
     *
     * @param $locale string - OPTIONAL defines the locale in which the day of the
week should be outputted
     * @param $format string - OPTIONAL defines the format in which the day of the
week should be outputted (1/sunday/...)
     * @return string
     */
    public function getDayOfYear($locale, $format) {}

    /**
```

```
     * Returns a defined calendar
     *
     * @param $calendar string - defines the type in which the calendar string should
be outputted (Gregorian, Islamic, Hebrew,...)
     * @param $locale string - defines the locale in which the calendar should be
outputted
     * @param $format string - defines the format in which the calendar should be
outputted
     * @return string
     */
    public function getCalendar($calendar, $locale, $format);

    /**
     * Sets a new date from calendar
     * Alias for set($calendar,Zend_Date::CALENDARFORMAT);
     *
     * @param $calendar string - date to set from a calendar string
     * @param $type string     - OPTIONAL type of calendar for parsing input
(Gregorian, Islamic, Hebrew,...)
     * @param $locale string   - OPTIONAL locale for parsing input
     * @param $format          - OPTIONAL an rule for parsing the input
     * @return object
     */
    public function setCalendar($calendar, $type, $locale, $format);

    /**
     * Adds a calendar date
     * Alias for add($calendar,Zend_Date::CALENDARFORMAT);
     *
     * @param $calendar string - date to add from a calendar string
     * @param $type string     - OPTIONAL type of calendar for parsing input
(Gregorian, Islamic, Hebrew,...)
     * @param $locale string   - OPTIONAL locale for parsing input
     * @param $format          - OPTIONAL an rule for parsing the input
     * @return object
     */
    public function addCalendar($calendar, $type, $locale, $format);

    /**
     * Substracts a calendar date
     * Alias for sub($calendar,Zend_Date::CALENDARFORMAT);
     *
     * @param $calendar string - date to sub from a calendar string
     * @param $type string     - OPTIONAL type of calendar for parsing input
(Gregorian, Islamic, Hebrew,...)
     * @param $locale string   - OPTIONAL locale for parsing input
     * @param $format          - OPTIONAL an rule for parsing the input
     * @return object
     */
    public function subCalendar($calendar, $type, $locale, $format);

    /**
     * Compares the calendar, returning the difference
     * Alias for compare($calendar,Zend_Date::CALENDARFORMAT);
     *
     * @param $calendar string - date to compare from a calendar string
     * @param $type string     - OPTIONAL type of calendar for parsing input
(Gregorian, Islamic, Hebrew,...)
     * @param $locale string   - OPTIONAL locale for parsing input
```

```
* @param $format          - OPTIONAL an rule for parsing the input
* @return object
*/
```

```
    public function compareCalendar($calendar, $type, $locale, $format);
}
```

REALISATION

To mention here:

The functions are not to realize as they were shown, instead almost all are aliases for general functions.
In real the realization will look like this :

```
    public function __call($function, $arguments) {
        switch($function) {
            "getHour" :
                $this->get(Zend_Date::HOUR,$arguments[0],$arguments[1]);
                break;
            "getMinute" :
                $this->get(Zend_Date::MINUTE,$arguments[0],$arguments[1]);
        }
    }
```

So the implementation itself will be very small.
Of course we could also do

```
    public function getHour($argument) {
        return $this->get(Zend_Date::HOUR,$arguments);
    }

    public function getMinute($arguments) {
        return $this->get(Zend_Date::MINUTE,$arguments);
    }
```

I think there's no difference in the speed.
]]></ac:plain-text-body></ac:macro>
]]></ac:plain-text-body></ac:macro>