Zend_Currency 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_Currency Component Proposal

Proposed Component Name	Zend_Currency
Developer Notes	http://framework.zend.com/wiki/display/ZFDEV/Zend_Currency
Proposers	Thomas Weidner Gavin (Zend-liaison)
Revision	2.0 - 18 July 2006: Changes because of comments from Zend 1.1 - 20 June 2006: Changed class to be static 1.0 - 20 June 2006: Initial release extracted from Zend_Locale and Zend_Unit. (wiki revision: 19)

Table of Contents

1. Overview
2. References
3. Component Requirements, Constraints, and Acceptance Criteria
4. Dependencies on Other Framework Components
5. Theory of Operation
Converting between locales/Unit conversion
Basically we have to implement/know the following standards
6. Milestones / Tasks
7. Class Index
8. Use Cases
9. Class Skeletons

1. Overview

Zend_Currency is a Class which handles all Currency related functions and is locale aware.

2. References

- Java Currency Object

3. Component Requirements, Constraints, and Acceptance Criteria

- Handling of currency related issues
- Works with locales

4. Dependencies on Other Framework Components

- Zend_Exception
- Zend_Locale_Data

5. Theory of Operation

Converting between locales/Unit conversion

Can output in different locales
Hint: Does not convert between Currencies (f.e. from Dollar to EUR!!

Basically we have to implement/know the following standards

ISO 4217 - Currency Codes

6. Milestones / Tasks

zone: Missing {zone-data:milestones}

7. Class Index

- Zend_Currency_Exception
- Zend_Currency

8. Use Cases

Get a currency string (standard locale)

```
$currency = new Zend_Currency('$123.4');
    echo $currency->getCurrency();   // outputs f.e. $123.4 when standard locale is en_US
```

Different inputs for a currency

```
$cur = new Zend_Currency('0');       // = $ 0
    $cur = new Zend_Currency('-123.4');  // = $ -123.4
    $cur = new Zend_Currency('12,345.678');  // = $ 12345.68    - standard precicion is 2 for en
    $cur = new Zend_Currency('I have 500 dollar and 40 cent');
    // value will be 500 as only the first value will be recognised
    $cur = new Zend_Currency(12,345.678');
    // when standard locale is en the value would be 12345.68
    // when standard locale is de the value would be 12.35
```

Get a currency string with different currency

```
$cur = new Zend_Currency('23,6',Zend_Locale::de_DE);
    echo $cur->getCurrency(Zend_Currency::EUR);
    // outputs 23.6 eur
```

Get a formatted currency string with different currency

```
$cur = new Zend_Currency('23.642');
    echo $cur->getCurrency(Zend_Currency::EUR,Zend_Locale_Format::NOPRECISION);
    // outputs 23 eur
```

Get a the standard currency for a spezified locale

```
$cur = new Zend_Currency('23.642');
    echo $cur->getCurrency(Zend_Currency::EUR,Zend_Locale_Format::NOPRECISION);
    // outputs 23 eur
    echo $cur->getCurrency(Zend_Locale::DE_DE);
    // outputs 23,642 eur
```

Get currency datas of a locale

```
echo Zend_Currency::getCurrencySign(Zend_Locale::DE_DE);  // outputs €
    echo Zend_Currency::getCurrencyName(Zend_Locale::DE_DE);  // outputs Euro
    echo Zend_Currency::getCurrencyShortName(Zend_Locale::DE_DE);  // outputs EUR
```

Maths with currencies

```
$cur1 = new Zend_Currency('$100');
    $cur2 = new Zend_Currency('$180');
    $cur = $cur1->add($cur2);
    echo $cur;  // outputs $ 280
    $cur = $cur1->sub($cur2);
    echo $cur;  // outputs $ -80
    $cur = $cur1->compare($cur2);
    echo $cur;  // outputs $ 80
```

9. Class Skeletons

```


class Zend_Currency_Exception extends Zend_Exception {}

class Zend_Currency implements Serializable {

    private $_currency = '';

    /**
     * Constructor
     *
     * @param $value mixed - can be integer, float or string
     * @param $locale      - OPTIONAL the locale for parsing
     */
    public function __construct($value, $locale) {}
```

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

    /**
     * Returns the parsed value of the currency
     *
     * @return mixed  - integer or float value
     */
    public function toValue() {}

    /**
     * Returns a complete currency string
     * Alias for toString($currency, OPTIONAL, OPTIONAL)
     *
     * @param $currency - new Currency Type, otherwise standard currency will be taken
     * @param $locale - OPTIONALLY the locale for formatting the currency
     * @param $format - OPTIONALLY formatting rules (f.e. precicion 2)
     * @return string
     */
    public function toCurrency($currency, $locale, $format) {}

    /**
     * Returns the currency symbol,
     * when no symbol is avaiable it returns the currency shortname (f.e. FIM for Finnian Mark)
     *
     * @param $type   - Currency Type
     * @param $locale - OPTIONALLY the locale for getting the sign
     * @return string
     */
    public static function getCurrencySign($type, $locale) {}

    /**
     * Returns the shortname of the currency
     *
     * @param $type   - Currency Type
     * @param $locale - OPTIONALLY the locale for getting the shortname
     * @return string
     */
    public static function getCurrencyShortName($type, $locale) {}

    /**
     * Returns the name of the currency
     *
     * @param $type   - Currency Type
     * @param $locale - OPTIONALLY the locale for getting the sign
     * @return string
     */
    public static function getCurrencyName($type, $locale) {}

    /**
     * Returns a list of regions where this currency is or was known
     *
     * @param $currency - Currency Type
     * @return array    - List of regions
     */
    public static function getRegionForCurrency($currency) {}

    /**
     * Returns a list of currencies which are used in this region
     *
     * @param $region - Currency Type
     * @return array  - List of currencys
     */
    public static function getCurrencyForRegion($region) {}

    /**
     * Returns the currency symbol,
```

```
     * when no symbol is avaiable it returns the currency shortname (f.e. FIM for Finnian Mark)
     *
     * @param $type   - Currency Type
     * @param $locale - OPTIONALLY the locale for getting the fraction
     * @return integer
     */
    public static function getFraction($type, $locale) {}

    /**
     * Returns the currency as string
     *
     * @param $currency - OPTIONAL other currency for output
     * @param $locale - OPTIONAL other locale for output (f.e. output in german locale 1.234,567)
     * @param $format - OPTIONAL other format for output (f.e. for rounding with precicion of 2
1,234,57)
     * @return string
     */
    public function toString($currency, $locale, $format);
    public function __toString($currency, $locale, $format);

    /**
     * Adds a currency to an other
     *
     * @param $currency mixed - currency object
     * @return currency - returns the new currency objects
     */
    public function add($currency) {}

    /**
     * Subtracts a currency from an other
     *
     * @param $currency mixed - currency object
     * @return currency - returns the new currency objects
     */
    public function sub($currency) {}

    /**
     * Compares two currencys, returning the difference
     *
     * @param $currency mixed - currency object
     * @return currency - returns the new currency objects
```

```
     */
    public function compare($currency) {}
}
```

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