Zend_Utf8 - Andrea Ercolino

<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_Utf8 Component Proposal

Proposed Component Name	Zend_Utf8
Developer Notes	http://framework.zend.com/wiki/display/ZFDEV/Zend_Utf8
Proposers	Andrea Ercolino
Zend Liaison	TBD
Revision	1.0 - 11 January 2011: Initial Draft. (wiki revision: 40)

Table of Contents

1. Overview
WordPress Plugin
2. References
3. Component Requirements, Constraints, and Acceptance Criteria
4. Dependencies on Other Framework Components
5. Theory of Operation
Options
6. Milestones / Tasks
7. Class Index
8. Use Cases
9. Class Skeletons

1. Overview

Zend_Utf8 is a simple component that offers escape and unescape functionalities. It's intended as a replacement for some code that is already available in ZF, though embedded in the Zend_Json and Zend_Serializer components. I've recently published a post about it at my site: http://noteslog.com/post/escaping-and-unescaping-utf-8-characters-in-php/

The Zend_Utf8 class is really simple, wholly coded, and ready for delivery, I hope. Note that still in the last release-1.11.2 the UTF-8 escaping feature in Zend_Json doesn't take into account all possible UTF-8 characters: in fact it lacks any support for the so called extended unicode characters, with a code point between 0×10000 and 0x10FFFF. This class does provide support for all unicode.

Encoding PHP values to some other string format, like JSON, could require escaping UTF-8 characters. It respectively goes for decoding and unescaping. I think it's sufficiently justified the existence of a class for basic UTF-8 support in the Zend Framework. When this class will be available, the Zend_Json and Zend_Serializer modules should be refactored to call Zend_Utf8 methods where needed.

WordPress Plugin

I've made a plugin for adding full UTF-8 support to WordPress. It's basically a wrapper of the class described here, which I have de-Zend-ified for distributing it in the wild.

2. References

- Escaping and unescaping UTF-8 characters in PHP
- UTF-8 and Unicode FAQ
- Zend_Json_Encoder
- Zend_Json_Decoder
- Zend_Serializer_Adapter_PythonPickle

3. Component Requirements, Constraints, and Acceptance Criteria

4. Dependencies on Other Framework Components

5. Theory of Operation

Zend_Utf8 exposes six static functions: two are the main functions for escaping and unescaping strings and four are the ancillary functions for mapping UTF-8 characters to unicode integers and the other way around. Usage of the ancillary functions is well documented by the main functions, so I'll describe only usage of the latter.

In the use cases I'm going to use the following functions:

```
function show($string, $options = array())
{
    $escape = Zend_Utf8::escape($string, $options);
    $unescape = Zend_Utf8::unescape($escape, $options);
    print_r("Original: $string\n");
    print_r("Escaped: $escape\n");
    print_r('-- escaped'  . ($string   === $unescape    ? " and "   : " BUT NOT ")
. 'unescaped as expected'."\n");
    if (empty($options))
    {
        $json_encode = trim(json_encode($string), '"');
        $json_decode = json_decode('"'.$escape.'"');
        print_r('-- escape'   . ($escape   === $json_encode ? " works " : " DOESN'T
WORK ") . 'like in json_encode'."\n");
        print_r('-- unescape' . ($unescape === $json_decode ? " works " : " DOESN'T
WORK ") . 'like in json_decode'."\n");
    }
    print_r("\n");
}
```

Options

Options are needed for changing the default behavior, and must be provided as an associative array.

key	type	description

extendedUseSurrogate	boolean	It controls how an extended unicode character will be represented: TRUE: a surrogate pair: write/read handlers will be called twice, each time receiving a member of the pair FALSE: a code point: write/read handlers will be called once, receiving the code point
write	handler	It controls how a code point (or each member of a surrogate pair) will be written to the escaped output.
read	handler	It controls how a code point (or each member of a surrogate pair) will be read from the escaped input.
filters	array	before-write: handler after-read: handler

A handler in the above table is an array with two keys: callback and arguments. Only the read handler has an additional key: a preg pattern to match against the escaped string for obtaining the string of an escaped value.

handler	callback input	callback output
write	arguments + unicode integer (code point or member of surrogate pair) + (unescaped) UTF-8 character	string of the escaped UTF-8 character
read	arguments + pattern matches	unicode integer (code point or member of surrogate pair)
before-write	arguments + unescaped string	unescape-safe string
after-read	arguments + unescape-safe string	unescaped string

6. Milestones / Tasks

- Milestone 1: [DONE] Working prototype
- Milestone 2: Unit tests exist, work, and are checked into SVN.
- Milestone 3: Initial documentation exists.

7. Class Index

- Zend_Utf8_Exception
- Zend_Utf8

8. Use Cases

9. Class Skeletons

Actually, these are class implementations.

```
class Zend_Utf8_Exception extends Zend_Exception
{}



/**
 * Basic UTF-8 support
 *
 * @link http://noteslog.com/
 * @link http://www.cl.cam.ac.uk/~mgk25/unicode.html#utf-8
 *
```

```
 * @package Zend_Utf8
 */
class Zend_Utf8
{
    /**
     * Escape UTF-8 characters using the given options
     *
     * About the write.callback option
     * -- it receives
     * -- -- the given write.arguments
     * -- -- the unicode of the current UTF-8 character
     * -- -- the current (unescaped) UTF-8 character
     * -- it must return the current escaped UTF-8 character
     *
     * @link http://noteslog.com/post/escaping-and-unescaping-utf-8-characters-in-php/
     *
     * @param  string $value
     * @param  array $options
     *   'escapeControlChars'   => boolean (default: TRUE),
     *   'escapePrintableASCII' => boolean (default: FALSE),
     *   'write'                => array(
     *       'callback'  => callable (default: 'sprintf'),
     *       'arguments' => array    (default: array('\u%04x')),
     *   ),
     *   'extendedUseSurrogate' => boolean (default: true),
     *
     * @throws Zend_Utf8_Exception If the code point of any char in $value is
     *                             not unicode
     * @return string
     */
    public static function escape($value, array $options = array())
    {
        $options = array_merge(array(
            'escapeControlChars'   => true,
            'escapePrintableASCII' => false,
            'write'                => array(
                'callback'  => 'sprintf',
                'arguments' => array('\u%04x'),
            ),
            'extendedUseSurrogate' => true,
        ), $options);
        if (! self::isCallable($options['write']))
        {
            require_once 'Zend/Utf8/Exception.php';
            throw new Zend_Utf8_Exception('Expected a valid write handler (callable,
array).');
        }
        if (self::validateFilters($options) &&
isset($options['filters']['before-write']))
        {
            $value = self::call($options['filters']['before-write'], $value);
        }

        $result = "";
        $length = strlen($value);
        for($i = 0; $i < $length; $i++) {
            $ord_var_c = ord($value[$i]);

            switch (true) {
```

```
                case ($ord_var_c < 0x20):
                    // code points 0x00000000..0x0000001F, mask 0xxxxxxx
                    $utf8Char = $value[$i];
                    $result .= $options['escapeControlChars']
                        ? self::call($options['write'], array($ord_var_c, $utf8Char))
                        : $value[$i];
                break;

                case ($ord_var_c < 0x80):
                    // code points 0x00000020..0x0000007F, mask 0xxxxxxx
                    $utf8Char = $value[$i];
                    $result .= $options['escapePrintableASCII']
                        ? self::call($options['write'], array($ord_var_c, $utf8Char))
                        : $value[$i];
                break;

                case (($ord_var_c & 0xE0) == 0xC0):
                    // code points 0x00000080..0x000007FF, mask 110yyyyy 10xxxxxx
                    $utf8Char = substr($value, $i, 2); $i += 1;
                    $code = self::utf8CharToCodePoint($utf8Char);
                    $result .= self::call($options['write'], array($code, $utf8Char));
                break;

                case (($ord_var_c & 0xF0) == 0xE0):
                    // code points 0x00000800..0x0000FFFF, mask 1110zzzz 10yyyyyy
10xxxxxx
                    $utf8Char = substr($value, $i, 3); $i += 2;
                    $code = self::utf8CharToCodePoint($utf8Char);
                    $result .= self::call($options['write'], array($code, $utf8Char));
                break;

                case (($ord_var_c & 0xF8) == 0xF0):
                    // code points 0x00010000..0x0010FFFF, mask 11110www 10zzzzzz
10yyyyyy 10xxxxxx
                    $utf8Char = substr($value, $i, 4); $i += 3;
                    if ($options['extendedUseSurrogate'])
                    {
                        list($upper, $lower) =
self::utf8CharToSurrogatePair($utf8Char);
                        $result .= self::call($options['write'], array($upper,
$utf8Char));
                        $result .= self::call($options['write'], array($lower,
$utf8Char));
                    }
                    else
                    {
                        $code = self::utf8CharToCodePoint($utf8Char);
                        $result .= self::call($options['write'], array($code,
$utf8Char));
                    }
                break;

                default:
                    //no more cases in unicode, whose range is 0x00000000..0x0010FFFF
                    require_once 'Zend/Utf8/Exception.php';
                    throw new Zend_Utf8_Exception('Expected a valid UTF-8
character.');
                break;
            }
```

```
        }

        return $result;
    }

    /**
     * Compute the code point of a given UTF-8 character
     *
     * If available, use the multibye string function mb_convert_encoding
     * TODO reject overlong sequences in $utf8Char
     *
     * @link http://noteslog.com/post/escaping-and-unescaping-utf-8-characters-in-php/
     *
     * @param  string $utf8Char
     * @throws Zend_Utf8_Exception If the code point of $utf8Char is not unicode
     * @return integer
     */
    public static function utf8CharToCodePoint($utf8Char)
    {
        if (function_exists('mb_convert_encoding'))
        {
            $utf32Char = mb_convert_encoding($utf8Char, 'UTF-32', 'UTF-8');
        }
        else
        {
            $bytes = array('C*');
            list(, $utf8Int) = unpack('N', str_repeat(chr(0), 4 - strlen($utf8Char)) .
$utf8Char);
            switch (strlen($utf8Char))
            {
                case 1:
                    //Code points U+0000..U+007F
                    //mask   0xxxxxxx (7 bits)
                    //map to 00000000 00000000 00000000 0xxxxxxx
                    $bytes[] = 0;
                    $bytes[] = 0;
                    $bytes[] = 0;
                    $bytes[] = $utf8Int;
                break;

                case 2:
                    //Code points U+0080..U+07FF
                    //mask   110yyyyy 10xxxxxx (5 + 6 = 11 bits)
                    //map to 00000000 00000000 00000yyy yyxxxxxx
                    $bytes[] = 0;
                    $bytes[] = 0;
                    $bytes[] = $utf8Int >> 10 & 0x07;
                    $bytes[] = $utf8Int >>  2 & 0xC0 | $utf8Int       & 0x3F;
                break;

                case 3:
                    //Code points U+0800..U+D7FF and U+E000..U+FFFF
                    //mask   1110zzzz 10yyyyyy 10xxxxxx (4 + 6 + 6 = 16 bits)
                    //map to 00000000 00000000 zzzzyyyy yyxxxxxx
                    $bytes[] = 0;
                    $bytes[] = 0;
                    $bytes[] = $utf8Int >> 12 & 0xF0 | $utf8Int >> 10 & 0x0F;
                    $bytes[] = $utf8Int >>  2 & 0xC0 | $utf8Int       & 0x3F;
                break;
```

```
                case 4:
                    //Code points U+10000..U+10FFFF
                    //mask   11110www 10zzzzzz 10yyyyyy 10xxxxxx (3 + 6 + 6 + 6 = 21
bits)
                    //map to 00000000 000wwwzz zzzzyyyy yyxxxxxx
                    $bytes[] = 0;
                    $bytes[] = $utf8Int >> 22 & 0x1C | $utf8Int >> 20 & 0x03;
                    $bytes[] = $utf8Int >> 12 & 0xF0 | $utf8Int >> 10 & 0x0F;
                    $bytes[] = $utf8Int >>  2 & 0xC0 | $utf8Int       & 0x3F;
                break;

                default:
                    //no more cases in unicode, whose range is 0x00000000 - 0x0010FFFF
                    require_once 'Zend/Utf8/Exception.php';
                    throw new Zend_Utf8_Exception('Expected a valid UTF-8
character.');
                break;
            }
            $utf32Char = call_user_func_array('pack', $bytes);
        }
        list(, $result) = unpack('N', $utf32Char); //unpack returns an array with base
1
        if (0xD800 <= $result && $result <= 0xDFFF)
        {
            //reserved for UTF-16 surrogates
            require_once 'Zend/Utf8/Exception.php';
            throw new Zend_Utf8_Exception('Expected a valid UTF-8 character.');
        }
        if (0xFFFE == $result || 0xFFFF == $result)
        {
            //reserved
            require_once 'Zend/Utf8/Exception.php';
            throw new Zend_Utf8_Exception('Expected a valid UTF-8 character.');
        }

        return $result;
    }

    /**
     * Compute the surrogate pair of a given extended UTF-8 character
     *
     * @link http://noteslog.com/post/escaping-and-unescaping-utf-8-characters-in-php/
     * @link http://en.wikipedia.org/wiki/UTF-16/UCS-2
     *
     * @param  string $utf8Char
     * @throws Zend_Utf8_Exception If the code point of $utf8Char is not extended
unicode
     * @return array
     */
    public static function utf8CharToSurrogatePair($utf8Char)
    {
        $codePoint = self::utf8CharToCodePoint($utf8Char);
        if ($codePoint < 0x10000)
        {
            require_once 'Zend/Utf8/Exception.php';
            throw new Zend_Utf8_Exception('Expected an extended UTF-8 character.');
        }
        $codePoint -= 0x10000;
```

```
        $upperSurrogate = 0xD800 + ($codePoint >> 10);
        $lowerSurrogate = 0xDC00 + ($codePoint & 0x03FF);
        $result = array($upperSurrogate, $lowerSurrogate);

        return $result;
    }

 /**
     * Unescape UTF-8 characters from a given escape format
     *
     * About the read.callback option
     * -- it receives
     * -- -- the given read.arguments
     * -- -- the current match of the pattern with all submatches
     * -- it must return the current unicode integer
     *
     * @link http://noteslog.com/post/escaping-and-unescaping-utf-8-characters-in-php/
     *
     * @param  string $value
     * @param  array $options
     *   'read'                 => array(
     *     'pattern'   => preg     (default: '@\\\\u([0-9A-Fa-f]{4})@'),
     *       'callback'  => callable (default: create_function('$all, $code', 'return
hexdec($code);')),
     *       'arguments' => array    (deafult: array()),
     *   ),
     *   'extendedUseSurrogate' => boolean (default: TRUE),
     *
     * @throws Zend_Utf8_Exception If the code point of any char in $value is
     *                             not unicode
     *
     * @return string
     */
    public static function unescape($value, array $options = array())
    {
        $options = array_merge(array(
         'read'                 => array(
             'pattern'   => '@\\\\u([0-9A-Fa-f]{4})@',
                'callback'  => create_function('$all, $code', 'return
hexdec($code);'),
                'arguments' => array(),
            ),
            'extendedUseSurrogate' => true,
        ), $options);

        if (! self::isCallable($options['read']))
        {
            require_once 'Zend/Utf8/Exception.php';
            throw new Zend_Utf8_Exception('Expected a valid read handler (callable,
array).');
        }
        $thereAreFilters = self::validateFilters($options);

        $result = "";
        while (preg_match($options['read']['pattern'], $value, $matches,
PREG_OFFSET_CAPTURE))
        {
            $unicode = self::eatUpMatches($result, $value, $matches,
$options['read']);
```

```
            if ($options['extendedUseSurrogate'] && (0xD800 <= $unicode && $unicode <
0xDC00))
            {
                $upperSurrogate = $unicode;
                if (! preg_match($options['read']['pattern'], $value, $matches,
PREG_OFFSET_CAPTURE))
                {
                    require_once 'Zend/Utf8/Exception.php';
                    throw new Zend_Utf8_Exception('Expected an extended UTF-8
character.');
                }
                $unicode = self::eatUpMatches($result, $value, $matches,
$options['read']);
                $utf8Char = self::utf8CharFromSurrogatePair(array($upperSurrogate,
$unicode));
            }
            else
            {
                $utf8Char = self::utf8CharFromCodePoint($unicode);
            }
            $result .= $utf8Char;
        }
        $result .= $value;

        if ($thereAreFilters && isset($options['filters']['after-read']))
        {
            $result = self::call($options['filters']['after-read'], $result);
        }

        return $result;
     }

    /**
     * Compute the UTF-8 character of a given code point
     *
     * If available, use the multibye string function mb_convert_encoding
     *
     * @link http://noteslog.com/post/escaping-and-unescaping-utf-8-characters-in-php/
     *
     * @param  integer $codePoint
     * @throws Zend_Utf8_Exception if the code point is not unicode
     * @return string
     */
    public static function utf8CharFromCodePoint($codePoint)
    {
        if (0xD800 <= $codePoint && $codePoint <= 0xDFFF)
        {
            //reserved for UTF-16 surrogates
            require_once 'Zend/Utf8/Exception.php';
            throw new Zend_Utf8_Exception('Expected a valid code point.');
        }
        if (0xFFFE == $codePoint || 0xFFFF == $codePoint)
        {
            //reserved
            require_once 'Zend/Utf8/Exception.php';
            throw new Zend_Utf8_Exception('Expected a valid code point.');
        }

        if (function_exists('mb_convert_encoding'))
```

```
        {
            $utf32Char = pack('N', $codePoint);
            $result = mb_convert_encoding($utf32Char, 'UTF-8', 'UTF-32');
        }
        else
        {
            $bytes = array('C*');
            switch (true)
            {
                case ($codePoint < 0x80):
                    //Code points U+0000..U+007F
                    //mask     0xxxxxxx (7 bits)
                    //map from xxxxxxx
                    $bytes[] = $codePoint;
                break;

                case ($codePoint < 0x800):
                    //Code points U+0080..U+07FF
                    //mask     110yyyyy 10xxxxxx (5 + 6 = 11 bits)
                    //map from yyy yyxxxxxx
                    $bytes[] = 0xC0 | $codePoint >> 6;
                    $bytes[] = 0x80 | $codePoint       & 0x3F;
                break;

                case ($codePoint < 0x10000):
                    //Code points U+0800..U+D7FF and U+E000..U+FFFF
                    //mask     1110zzzz 10yyyyyy 10xxxxxx (4 + 6 + 6 = 16 bits)
                    //map from zzzzyyyy yyxxxxxx
                    $bytes[] = 0xE0 | $codePoint >> 12;
                    $bytes[] = 0x80 | $codePoint >> 6  & 0x3F;
                    $bytes[] = 0x80 | $codePoint       & 0x3F;
                break;

                case ($codePoint < 0x110000):
                    //Code points U+10000..U+10FFFF
                    //mask     11110www 10zzzzzz 10yyyyyy 10xxxxxx (3 + 6 + 6 + 6 = 21
bits)
                    //map from wwwzz zzzzyyyy yyxxxxxx
                    $bytes[] = 0xF0 | $codePoint >> 18;
                    $bytes[] = 0x80 | $codePoint >> 12 & 0x3F;
                    $bytes[] = 0x80 | $codePoint >> 6  & 0x3F;
                    $bytes[] = 0x80 | $codePoint       & 0x3F;
                break;

                default:
                    require_once 'Zend/Utf8/Exception.php';
                    throw new Zend_Utf8_Exception('Expected a valid code point.');
                break;
            }
            $result = call_user_func_array('pack', $bytes);
        }
        return $result;
    }

    /**
     * Compute the extended UTF-8 character of a given surrogate pair
     *
     * @link
http://noteslog.com/post/escaping-and-unescaping-utf-8-characters-in-php/
```

```
     * @link http://en.wikipedia.org/wiki/UTF-16/UCS-2
     *
     * @param array $surrogatePair
     * @throws Zend_Utf8_Exception If the surrogate pair is not extended unicode
     * @return string
     */
    public static function utf8CharFromSurrogatePair($surrogatePair)
    {
        list($upperSurrogate, $lowerSurrogate) = $surrogatePair;
        if (! (0xD800 <= $upperSurrogate && $upperSurrogate < 0xDC00))
        {
            require_once 'Zend/Utf8/Exception.php';
            throw new Zend_Utf8_Exception('Expected an extended UTF-8 character.');
        }
        if (! (0xDC00 <= $lowerSurrogate && $lowerSurrogate < 0xE000))
        {
            require_once 'Zend/Utf8/Exception.php';
            throw new Zend_Utf8_Exception('Expected an extended UTF-8 character.');
        }
        $codePoint = ($upperSurrogate & 0x03FF) << 10 | ($lowerSurrogate & 0x03FF);
        $codePoint += 0x10000;
        $result = self::utf8CharFromCodePoint($codePoint);

        return $result;
    }

    /**
     * Validate filters. If there are filters return true, else false
     *
     * @param array $options
     * @throws Ando_Utf8_Exception If there are malformed filters
     * @return boolean
     */
    protected static function validateFilters($options)
    {
        if (isset($options['filters']))
        {
            if (! is_array($options['filters']))
            {
                require_once 'Zend/Utf8/Exception.php';
                throw new Zend_Utf8_Exception('Expected valid filters.');
            }
            foreach ($options['filters'] as $key => $value)
            {
                if (! self::isCallable($value))
                {
                    require_once 'Zend/Utf8/Exception.php';
                    throw new Zend_Utf8_Exception("Expected a valid $key handler.");
                }
            }
            return true;
        }
        return false;
    }

    /**
     * A little calling interface: validation
     *
     * @param  array  $handler
```

```
     * @return boolean
     */
    private static function isCallable($handler)
    {
        $result = is_callable($handler['callback']) &&
is_array($handler['arguments']);
        return $result;
    }

    /**
     * A little calling interface: call
     *
     * @param  array  $handler
     * @param  mixed  $args
     * @return mixed
     */
    private static function call($handler, $args)
    {
        $args = array_merge($handler['arguments'], is_array($args) ? $args :
array($args));
        $result = call_user_func_array($handler['callback'], $args);
        return $result;
    }

    /**
     * Return the transposition of the given array
     *
     * @param array $rows
     * @return array
     */
    private static function transpose($rows)
    {
        $result = call_user_func_array('array_map', array_merge(array(null), $rows));
        return $result;
    }

    /**
     * 1: update $processed with the unmatched substring before $matches
     * 2: update $value with the rest of the substring after $matches
     * 3: return unicode read from the matched substring in $matches
     *
     * @param string $processed
     * @param string $value
     * @param array $matches
     * @param array $handler
     * @return integer
     */
    private static function eatUpMatches(&$processed, &$value, $matches, $handler)
    {
        $match  = $matches[0][0];
        $offset = $matches[0][1];
        $processed .= substr($value, 0, $offset);
        $value = substr($value, $offset + strlen($match));

        $matches = self::transpose($matches);
        $args = $matches[0];
        $result = self::call($handler, $args);

        return $result;
```

```
}
```

```
}
```

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