Zend_Uri 2.0 - Shahar Evron

<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_Uri Component Proposal

Proposed Component Name	Zend_Uri
Developer Notes	http://framework.zend.com/wiki/display/ZFDEV/Zend_Uri
Proposers	Shahar Evron
Zend Liaison	TBD
Revision	1.0 - 2 August 2010: Ready for community review 0.1 - 26 July 2010: Initial Draft (wiki revision: 40)

Table of Contents

1. Overview
2. References
3. Component Requirements, Constraints, and Acceptance Criteria
4. Dependencies on Other Framework Components
5. Theory of Operation
Subclassing
Parsing and Composition
Relative URI Resolution
Normalization
Automatic Scheme-specific Class Selection
Scheme-specific Functionality
6. Milestones / Tasks
7. Class Index
8. Use Cases
9. Class Skeletons

1. Overview

Zend_Uri is the Zend Framework component responsible for representing URIs (Uniform Resource Identifiers) as objects.

In Zend Framework 1.x and before, Zend_Uri was mostly used for representation and validation of URIs of specific schemes, and the only scheme implemented was of HTTP URIs. In addition, Zend_Uri 1.0 was not capable of representing partial or relative URIs, and URIs of arbitrary schemes, and did not provide tools for resolving, encoding and normalizing URIs.

This proposal describes a set of changes and improvements (effectively a complete rewrite) to Zend_Uri for Zend Framework 2.0, that will address the deficiencies mentioned above.

2. References

- RFC 3986 - Uniform Resource Identifier (URI): Generic Syntax
- RFC 2616 Section 3.2 - Uniform Resource Identifiers
- uriparser - A C/C++ library for Generic Syntax URI parsing

- IANA URI scheme registry - list of registered URI schemes and RFCs defining them
- Preview code is available on the zend-uri-se branch at git://arr.gr/zf2.git

3. Component Requirements, Constraints, and Acceptance Criteria

- Zend\Uri **will** allow representation of generic URIs as objects
- Zend\Uri **will** allow programmatic composition of URIs using getter/setter methods to various URI parts
- Zend\Uri **will** allow creation of URI objects through string parsing
- Zend\Uri **will** allow representation of partial and relative URIs
- Zend\Uri **will** closely follow RFC-3986 definitions for URI syntax
- Zend\Uri **will** always produce RFC-3986 compliant URIs when converting URI objects back to a string
- Zend\Uri **will** attempt to be flexible when parsing string URIs and accept invalid URIs or URI parts if these can be encoded into a syntactically valid URI
- Zend\Uri **will** provide subclasses for representation of scheme-specific URIs
- Zend\Uri **will** allow users to easily create their own scheme-specific classes
- Zend\Uri scheme subclasses **may** enforce additional validation rules on URIs
- Zend\Uri scheme subclasses **may** provide additional protocol or scheme specific APIs
- Zend\Uri **will** allow automatic string-to-object conversion using the factory pattern
- Zend\Uri **will** provide API for resolving relative URIs
- Zend\Uri **will** provide API for normalizing URI strings
- Zend\Uri **will** provide API for converting absolute URIs into relative URIs based on a shared absolute base URI
- Zend\Uri **will** provide generic methods for validating and encoding different URI parts
- Zend\Uri **will not** provide an interface for strict validation or encoding of URI strings.
    - These operations should be provided by Zend\Validate and Zend\Filter classes that may internally rely on Zend\Uri encoding and validation methods.

4. Dependencies on Other Framework Components

- Zend\Validate\Hostname
- Zend\Validate\Ip
- Zend\Exception

5. Theory of Operation

Subclassing

The Zend\Uri component will provide a concrete class (Zend\Uri\Uri) implementing RFC-3986 compatible Generic URI Syntax parsing, composition, resolution, validation encoding and normalization of URIs. This class will be concrete and could be used to represent any compliant URI, including scheme specific URIs and partial or relative URIs.

In addition, Zend\Uri will provide a set of subclasses of Zend\Uri\Uri (initially Zend\Uri\Http and Zend\Uri\File) that will only be capable of representing URIs of specific schemes, and will enforce additional validation rules in addition to those defined by the Generic Syntax RFC. These subclasses may still be able to represent partial or relative URIs, as long as they comply with any rules imposed by the scheme.

Parsing and Composition

URI parsing and composition will be done following the parsing and composition rules defined in the RFC. The aim is to be relatively lax when parsing string URIs and setting different URI parts using accessor methods, and accept input as long as it can eventually be encoded into a valid URI when the object is converted back to a string.

For example, the URI `file:///C:/Program Files/Zend` will be accepted by the parser despite the fact that it's path component (C:/Program Files/Zend) contains an invalid space character. When the URI is composed back into a string and is normalized, it will be represented as `file:///C:/Program%20Files/Zend`, which is a valid and RFC-compliant URI.

Zend\Uri will refuse to parse a string or accept a part set through one of the mutator methods only if the input can never be unambiguously converted into a valid URI part.

For example, the following will not be allowed:

```
$uri->setScheme('my scheme');
```

Since the scheme of a URI may never contain spaces and the URI syntax rules do not define a mean to represent a space character in the scheme part.

In contrast, the following will be allowed:

```
$uri->setQuery('pound sign=#');
```

Since although the ' ' and '#' signs may not be used literally in the query part of a URI, they can be encoded as '%20' and '%23' respectively when the URI is re-composed.

Relative URI Resolution

One of the common tasks to preform with URIs is resolving relative URIs and merging a base URI with a relative URI to form a canonical representation of the relative URI. Unlike Zend_Uri 1.0, the new implementation will expose an API for resolving a (possibly relative) URI against an absolute base URI to form an absolute URI.

Additionally, Zend\Uri will expose an API to perform the opposite operation: "subtract" a common base URI from an absolute URI to form a relative reference.

Both methods can be useful for example when composing or parsing HTML pages, and when creating links in portable applications.

Normalization

Zend\Uri\Uri and it's subclasses will expose an API to normalize URI objects. This normalization method should be used, for example, before comparing two URI strings to check if they are identical.

For example, the following URLs, while syntactically different, are semantically equivalent: http://www.example.com:80/?foo=b%61 HTTP://www.example.com?foo=bar

The normalization API will allow the user to compare these two URIs, by normalizing them using the RFC defined normalization rules (and possibly scheme-specific normalization added in Zend\Uri\Uri subclasses). In the example above, the normalized URIs would both be converted to: http://www.example.com/?foo=bar

Normalizing URIs will include:

- Converting the scheme to lower case
- Removing the port if it is equal to the scheme's default port
- Decoding any percent-encoded characters which do not need to be encoded
- Replacing an empty path with '/' in URIs that have an authority part
- Converting percent-encoding hexadecimal characters to upper case
- Removing empty port, query or fragment parts
- Additional scheme specific normalization (e.g. in HTTP URLs lower-casing the host name)

Automatic Scheme-specific Class Selection

Zend\Uri will provide a factory-pattern class (Zend\Uri\UriFactory) which will allow users to pass URI strings into it. Depending on the URI string scheme, the Factory method will return a scheme-specific class to represent the URI if such class is registered with the UriFactory class. By

default, the scheme-specific classes provided by Zend\Uri will be registered with the factory class, and users will be able to register additional scheme-specific subclasses of Zend\Uri (or overwrite any pre-registered schemes) with the factory class, if they wish to implement their own URI classes.

If the URI string is relative and does not specify a scheme, users may specify a default scheme to fall back to (e.g. when parsing an HTML page fetched using HTTP, relative `href` links are assumed to be of the HTTP scheme).

If the factory method will not find the appropriate scheme class, or is unable to detect the URI scheme and no default scheme was specified, it will fall back to using the generic syntax Zend\Uri\Uri class.

Scheme-specific Functionality

In addition to the above, and to the enforcement of specific validation rules, Zend\Uri\Uri subclasses may also expose additional scheme-specific functionality.

For example, the File scheme class may expose methods to convert a Win32 or Unix file path into a file:// URI.

6. Milestones / Tasks

The implementation and general availability of this change should be coordinated with the release schedule of Zend Framework 2.0, with the following milestones:

- Milestone 1: [PARTIALLY DONE] design notes will be published here
- Milestone 2: [PARTIALLY DONE] A working prototype with 80%+ unit test coverage is checked into the developer's git repository
- Milestone 3: Working prototypes of scheme specific classes are checked into the developer's git repository
- Milestone 4: Complete unit test coverage and documentation is checked into the developer's git repository
- Milestone 5: Fixes to components relying on Zend_Uri (namely Zend\Http\Client) is checked into the developer's git repository
- Milestone 6: Zend\Filter and Zend\Validate classes for URI normalization and validation are made available
- Milestone 7: Developer branch is merged into the public git repository in time for preview releases of Zend Framework 2.0

7. Class Index

- Zend\Uri\Uri
- Zend\Uri\Http
- Zend\Uri\File
- Zend\Uri\UriFactory

Exception classes:

- Zend\Uri\Exception
- Zend\Uri\InvalidUriException
- Zend\Uri\InvalidUriPartException
- Zend\Uri\InvalidUriClassException

Filter / Validator implementations (optional):

- Zend\Filter\Uri - Zend Filter for normalizing URI strings
- Zend\Validator\Uri - Zend Validator for validating URIs

8. Use Cases

UC-01: URI parsing

```
// Using the class constructor
$uri = new Zend\Uri\Uri('scheme://foo.bar/baz');

// Using the parse() method
$uri = new Zend\Uri\Http;
$uri->parse('https://foo.bar/baz');

// This will throw an exception for trying to parse an http URI using a File object
$uri = new Zend\Uri\File;
$uri->parse('http://baz.com');
```

UC-02: Accessing different URI parts

```
/**
 * Setting URI parts through mutator methods
 */
$uri = new \Zend\Uri\Uri('/foo/bar.txt');
$uri->setQuery('baz=bar');
$uri->setFragment('myFragment rocks!');
echo $uri; // URI is strigified using __toString()
// Output: /foo/bar.txt?baz=bar#myFragment%20rocks%21

/**
 * Accessing URI parts
 */
echo $uri->getFragment();
// Output: myFragment rocks!

var_dump($uri->getScheme());
// Output: NULL

var_dump($uri->getQueryAsArray());
// Output:
// array(1) {
//  ["baz"]=>
//  string(3) "bar"
// }
```

UC-03: Resolving relative URIs

```
use \Zend\Uri\Uri;
// Either a string or a Uri object will work...
$baseUri = 'http://www.example.com/foo/bar?baz'

// The ->toString() method converts the URI into a string, equivalent to calling __toString()
$uri = new Uri('#banana');
echo $uri->resolve($baseUri)->toString();
// Output: http://www.example.com/foo/bar?baz#banana

$uri = new Uri('waka.html'); // ./waka.html will also do the same
echo $uri->resolve($baseUri)->toString();
// Output: http://www.example.com/foo/waka.html

$uri = new Uri('../file.txt?query');
echo $uri->resolve($baseUri)->toString();
// Output: http://www.example.com/file.txt?query

$uri = new Uri('http://www.otherdomain.com/file');
echo $uri->resolve($baseUri)->toString();
// Output: http://www.otherdomain.com/file
// The original URI is absolute so resolving has no effect
```

UC-04: Creating configuration-based URIs depending on environment

```
/**
 * Bootstrap.php: Bootstrap file
 *
 * application.ini contains the following settings:
 *
 * [production]
 * resources.uri.yui    = 'http://yui.yahooapis.com/2.8.0/build/'
 * resources.uri.images = 'http://static.example.com/files/images/'
 *
 * [development:production]
 * resources.uri.yui    = '/js/yui/2.8.0/'
 * resources.uri.images = '/images/'
 */

$config = new \Zend\Config\Ini('application.ini', APPLICATION_ENV);

$baseUrls = new \Zend\Registry(array(
    'yui'    => $config->resources->uri->yui,
    'images' => $config->resources->uri->images
));

\Zend\Registry::set('urlRegistry', $baseUrls);
```

```
/**
 * layout.phtml - Layout view script
 */

$urlRegistry = \Zend\Registry::get('urlRegistry');

$logoUrl     = $urlRegistry->images->merge('logo.png')->toString();
$gridCssUrl  = $urlRegistry->yui->merge('grids/grids-min.css')->toString();

/**
 * Full URLs can now be used in view script
 *
 * $logoUrl is now:
 *   development: '/images/logo.png'
 *   production:  'http://static.example.com/files/images/logo.png'
 *
 * $gridCssUrl is now:
 *   development: '/js/yui/2.8.0/grids/grids-min.css'
 *   production:  'http://yui.yahooapis.com/2.8.0/build/grids/grids-min.css'
 */
```

UC-05: Normalizing URIs

```
use \Zend\Uri;

$uri1 = new Uri\Http('http://myhost.com:80/foo/bar?url=http%3A%2F%2Fwww.example.com%2Fbaz');
$uri2 = new Uri\Http('HTTP://MYHOST.COM/foo/bar?url=http://www.example.com/baz');

echo $uri1;
// Output: http://myhost.com:80/foo/bar?url=http%3A%2F%2Fwww.example.com%2Fbaz

echo $uri2;
// Output: HTTP://MYHOST.COM/foo/bar?url=http://www.example.com/baz

// URLs are not equal
echo $uri1 == $uri2 ? 'yes' : 'no';
// Output: no

// Normalize both URLs
$uri1->normalize();
$uri2->normalize();

echo $uri1 == $uri2 ? 'yes' : 'no';
// Output: yes

echo $uri1 . "\n" . $uri2;
// Output:
//   http://myhost.com/foo/bar?url=http://www.example.com/baz
//   http://myhost.com/foo/bar?url=http://www.example.com/baz
```

UC-06: Creating a relative reference from an absolute URI

```
use Zend\Uri\Uri;

$baseUri = new Uri('http://example.com/dir/subdir/');

$file1 = new Uri('http://example.com/dir/subdir/more/file1.txt');
$file2 = new Uri('http://example.com/dir/otherdir/file2.txt');
$file3 = new Uri('http://otherhost.com/dir/subdir/file3.txt');

echo $file1->makeRelative($baseUri);
// Output: more/file1.txt

echo $file2->makeRelative($baseUri);
// Output: ../otherdir/file2.txt

echo $file3->makeRelative($baseUri);
// Output: 'http://otherhost.com/dir/subdir/file3.txt'
// (two URLs have no common parts)
```

UC-07: Extracing HTTP URLs from an HTML page

```
$baseUrl = new Uri\Http('http://www.example.com/base/url');

// ... fetch $baseUrl html page using Http\Client, and extract all <a href> links to $htmlLinks ...

$links = array();
foreach($htmlLinks as $link) {
  $scheme = \Zend\Uri\Uri::parseScheme($link);
  if ($scheme == 'http' || $scheme == 'https' || $scheme === null) {
    try {
      $url = new \Zend\Uri\Http($link);
      $links[] = $url->resolve($baseUrl)->normalize();
    } catch (\Zend\Uri\InvalidUriException $ex) {
      // Deal with invalid / malformed URI
    }
  }
}
// $links now contain all HTTP links from the page, normalized, with relative links resolved using the
original base URL.
```

UC-08: Using the Factory class with custom URI classes

```
// Registering a new scheme
/**
 * Subclass of Zend\Uri\Uri to represent files that can be accessed over SFTP
 */
class \MyFramework\Uri\Sftp extends \Zend\Uri\Uri
{
  static protected $_validSchemes = array('sftp');
}

\Zend\Uri\UriFactory::registerScheme('sftp', '\MyFramework\Uri\Sftp');
$uri = \Zend\Uri\UriFactory::factory($sftpUri);

// Overriding a default scheme class with a custom implementation
\Zend\Uri\UriFactory::registerScheme(array('http', 'https'), '\MyFramework\Uri\Http');
$uri = \Zend\Uri\UriFactory::factory('http://www.example.com/');
// $uri is now an instance of \MyFramework\Uri\Http
```

UC-09: Example of using Zend\Uri to get files from FTP server

```
use Zend\Uri\Uri;

// Parse URI
$ftpUri = new Uri('ftp://annon:mypass@ftp.example.com/pub/files/');
$files = array('file1.txt', 'file2.txt', 'file3.txt');

// Connect to server
$conn = ftp_connect($ftpUri->getHost(), $ftpUri->getPort());
if ($userInfo = $ftpUri->getUserInfo()) {
    list($username, $password) = explode(':', $userInfo);
    ftp_login($conn, $username, $password)
}

// Get the files
foreach($files as $file) {
    $path = new Uri($file);
    $path->resolve($ftpUri->getPath());
    ftp_get($conn, $file, $path->getPath(), FTP_BINARY);
}
```

UC-10: Accessing and Encoding various HTTP URI parts (part of an HTTP client implementation)

```
use Zend\Uri\Http as HttpUri;

function \httpGet(HttpPUri $url)
{
    // Check that we have a avalid absolute URL
    if (! ($url->isValid() && $url->isAbsolute()) {
        throw new \ErrorException("'$url' is not a valid, absolute HTTP URL");
    }

    // Normalize the URL before doing anything
    $url->normalize();

    // Connection params vary depending on whether HTTPS or HTTP are used
    // If getPort() returns NULL (or anything that evaluates to FALSE), use the default prot
    if ($url->getScheme() == 'https') {
        $host = 'ssl://' . $url->getHost();
        $port = ($url->getPort() ?: 443);
    } else {
        $host = $url->getHost();
        $port = ($url->getPort() ?: 80);
    }

    // For the purpose of this example we will avoid error checking...
    $conn = fsockopen($host, $port);

    // Path and query must be encoded before sending to the server
    $path = HttpUri::encodePath($url->getPath());
    if ($url->getQuery()) {
        $path .= '?' . HttpUri::encodeQueryFragment($url->getQuery());
    }

    $request = "GET $path HTTP/1.0\r\n" .
               "Host: {$url->getHost()}\r\n" .
               "Connection: close\r\n" .
               "\r\n";

    fwrite($conn, $request);

    // .. continue to read response, close connection etc.
}
```

9. Class Skeletons

The Zend\Uri\Uri class

```
namespace Zend\Uri;

/**
 * Generic URI handler
 *
 * @uses      \Zend\Uri\Exception
 * @uses      \Zend\Validator\Hostname
 * @uses      \Zend\Validator\Ip
 */
class Uri
{
    /**
     * Host part address types
     */
    const HOST_IPV4      = 1;   // IPv4 address
    const HOST_IPV6      = 2;   // IPv6 address
    const HOST_IPVF      = 4;   // IPvx (future) address
    const HOST_IPVANY    = 7;   // Any IP address
```

```
    const HOST_DNSNAME   = 8;   // DNS name
    const HOST_DNSORIPV4 = 9;   // DNS or IPv4
    const HOST_REGNAME   = 16;  // Registered name
    const HOST_ALL       = 31;  // Any address type

    /**
     * Character classes defined in RFC-3986
     */
    const CHAR_UNRESERVED = '\w\-\.~';
    const CHAR_GEN_DELIMS = ':\/\?#\[\]@';
    const CHAR_SUB_DELIMS = '!\$&\'\(\)\*\+,;=';
    const CHAR_RESERVED   = ':\/\?#\[\]@!\$&\'\(\)\*\+,;=';

    /**
     * Create a new URI object
     *
     * @param  \Zend\Uri\Uri | string | null $uri
     * @throws \InvalidArgumentException
     */
    public function __construct($uri = null);

    /**
     * Check if the URI is valid
     *
     * Note that a relative URI may still be valid
     *
     * @return boolean
     */
    public function isValid();

    /**
     * Check if the URI is an absolute or relative URI
     *
     * @return boolean
     */
    public function isAbsolute();

    /**
     * Parse a URI string
     *
     * @return \Zend\Uri\Uri
     */
    public function parse($uri);

    /**
     * Compose the URI into a string
     *
     * @return string
     */
    public function toString();

    /**
     * Normalize the URI
     *
     * Normalizing a URI includes removing any redundant parent directory or
     * current directory references from the path (e.g. foo/bar/../baz becomes
     * foo/baz), normalizing the scheme case, decoding any over-encoded
     * characters etc.
     *
     * Eventually, two normalized URLs pointing to the same resource should be
     * equal even if they were originally represented by two different strings
     *
     * @return \Zend\Uri\Uri
     */
    public function normalize();

    /**
     * Merge a relative URI into the current (usually absolute) URI, using the
     * current URI as a base reference point.
     *
```

```
     * Merging algorithm is adapted from RFC-3986 section 5.2
     * (@link http://tools.ietf.org/html/rfc3986#section-5.2)
     *
     * @param  \Zend\Uri\Uri | string $refUri
     * @return \Zend\Uri\Uri
     */
    public function merge($refUri);

    /**
     * Convert a relative URI into an absolute URI using a base absolute URI as
     * a reference.
     *
     * @param  \Zend\Uri\Uri | string $baseUrl
     * @return \Zend\Uri\Uri
     */
    public function resolve($baseUrl);

    /**
     * Convert the link to a relative link by substracting a base URI
     *
     *  This is the opposite of resolving a relative link - i.e. creating a
     *  relative reference link from an original URI and a base URI.
     *
     *  If the two URIs do not intersect (e.g. the original URI is not in any
     *  way related to the base URI) the URI will not be modified.
     *
     * @param  \Zend\Uri\Uri | string $baseUrl
     * @return \Zend\Uri\Uri
     */
    public function makeRelative($baseUrl);

    /**
     * Get the scheme part of the URI
     *
     * @return string | null
     */
    public function getScheme();

    /**
     * Get the User-info (usually user:password) part
     *
     * @return string | null
     */
    public function getUserInfo();

    /**
     * Get the URI host
     *
     * @return string | null
     */
    public function getHost();

    /**
     * Get the URI port
     *
     * @return integer | null
     */
    public function getPort();

    /**
     * Get the URI path
     *
     * @return string | null
     */
    public function getPath();

    /**
     * Get the URI query
     *
     * @return string | null
```

```
     */
    public function getQuery();

    /**
     * Return the query string as an associative array of key => value pairs
     *
     * This is an extension to RFC-3986 but is quite useful when working with
     * most common URI types
     *
     * @return array
     */
    public function getQueryAsArray();

    /**
     * Get the URI fragment
     *
     * @return string | null
     */
    public function getFragment();

    /**
     * Set the URI scheme
     *
     * If the scheme is not valid according to the generic scheme syntax or
     * is not acceptable by the specific URI class (e.g. 'http' or 'https' are
     * the only acceptable schemes for the Zend\Uri\HTTTP class) an exception
     * will be thrown.
     *
     * You can check if a scheme is valid before setting it using the
     * validateScheme() method.
     *
     * @param  string $scheme
     * @throws \Zend\Uri\InvalidSchemeException
     * @return \Zend\Uri\Uri
     */
    public function setScheme($scheme);

    /**
     * Set the URI User-info part (usually user:password)
     *
     * @param  string $userInfo
     * @return \Zend\Uri\Uri
     */
    public function setUserInfo($userInfo);

    /**
     * Set the URI host
     *
     * Note that the generic syntax for URIs allows using host names which
     * are not neceserily IPv4 addresses or valid DNS host names. For example,
     * IPv6 addresses are allowed as well, and also an abstract "registered name"
     * which may be any name composed of a valid set of characters, including,
     * for example, tilda (~) and underscore (_) which are not allowed in DNS
     * names.
     *
     * Subclasses of \Zend\Uri\Uri may impose more strict validation of host
     * names - for example the HTTP RFC clearly states that only IPv4 and
     * valid DNS names are allowed in HTTP URIs.
     *
     * @param  string $host
     * @return \Zend\Uri\Uri
     */
    public function setHost($host);

    /**
     * Set the port part of the URI
     *
     * @param  integer $port
     * @return \Zend\Uri\Uri
     */
```

```
    public function setPort($port);

    /**
     * Set the path
     *
     * @param  string $path
     * @return \Zend\Uri\Uri
     */
    public function setPath($path);

    /**
     * Set the query string
     *
     * @param  string | array $query
     * @return \Zend\Uri\Uri
     */
    public function setQuery($query);

    /**
     * Set the URI fragment part
     *
     * @param  string $fragment
     * @return \Zend\Uri\Uri
     */
    public function setFragment($fragment);

    /**
     * Magic method to convert the URI to a string
     *
     * @return string
     */
    public function __toString();

    /**
     * Encoding and Validation Methods
     */

    /**
     * Check if a scheme is valid or not
     *
     * Will check $scheme to be valid against the generic scheme syntax defined
     * in RFC-3986. If the class also defines specific acceptable schemes, will
     * also check that $scheme is one of them.
     *
     * @param  string $scheme
     * @return boolean
     */
    static public function validateScheme($scheme);

    /**
     * Check that the userInfo part of a URI is valid
     *
     * @param string $userInfo
     * @return boolean
     */
    static public function validateUserInfo($userInfo);

/**
     * Validate the host part
     *
     * Users may control which host types to allow by passing a second parameter
     * with a bitmask of HOST_* constants which are allowed. If not specified,
     * all address types will be allowed.
     *
     * Note that the generic URI syntax allows different host representations,
     * including IPv4 addresses, IPv6 addresses and future IP address formats
     * enclosed in square brackets, and registered names which may be DNS names
     * or even more complex names. This is different (and is much more loose)
     * from what is commonly accepted as valid HTTP URLs for example.
     *
```

```
     * @param  string  $host
     * @param  integer $allowed bitmask of allowed host types
     * @return boolean
     */
    static public function validateHost($host, $allowed = self::HOST_ALL);

    /**
     * Validate the port
     *
     * Valid values include numbers between 1 and 65535, and empty values
     *
     * @param  integer $port
     * @return boolean
     */
    static public function validatePort($port);

    /**
     * Validate the path
     *
     * @param  string $path
     * @return boolean
     */
    static public function validatePath($path);

    /**
     * Check if a URI query or fragment part is valid or not
     *
     * Query and Fragment parts are both restricted by the same syntax rules,
     * so the same validation method can be used for both.
     *
     * You can encode a query or fragment part to ensure it is valid by passing
     * it through the encodeQueryFragment() method.
     *
     * @param  string $input
     * @return boolean
     */
    static public function validateQueryFragment($input);

    /**
     * URL-encode the user info part of a URI
     *
     * @param  string $userInfo
     * @return string
     */
    static public function encodeUserInfo($userInfo);

    /**
     * Encode the path
     *
     * @param  string $path
     * @return string
     */
    static public function encodePath($path)
    {
        throw new Exception("Implelemt me!");
    }

    /**
     * URL-encode a query string or fragment based on RFC-3986 guidelines.
     *
     * Note that query and fragment encoding allows more unencoded characters
     * than the usual rawurlencode() function would usually return - for example
     * '/' and ':' are allowed as literals.
     *
     * @param  string $input
     * @return string
     */
    static public function encodeQueryFragment($input);

    /**
```

```
 * Remove any extra dot segments (/../, /./) from a path
 *
 * Algorithm is adapted from RFC-3986 section 5.2.4
 * (@link http://tools.ietf.org/html/rfc3986#section-5.2.4)
 *
 * @param  string $path
 * @return string
```

```
     */
    static public function removePathDotSegments($path);
}
```

```
/**
 * @namespace
 */
namespace Zend\Uri;
use \Zend\Uri\Uri;

/**
 * HTTP URI handler
 *
 * @uses      \Zend\Uri\Uri
 * @uses      \Zend\Uri\Exception
 * @category  Zend
 * @package   Zend_Uri
 * @copyright Copyright (c) 2005-2010 Zend Technologies USA Inc. (http://www.zend.com)
 * @license   http://framework.zend.com/license/new-bsd     New BSD License
 */
class Http extends Uri
{
    static protected $_validSchemes = array('http', 'https');

    /**
     * Check if the URI is a valid HTTP URI
     *
     * This applys additional HTTP specific validation rules beyond the ones
     * required by the generic URI syntax
     *
     * @return boolean
     * @see    \Zend\Uri\Uri::isValid()
     */
    public function isValid();

    /**
     * Get the username part (before the ':') of the userInfo URI part
     *
     * @return string
     */
    public function getUser();

    /**
     * Get the password part (after the ':') of the userInfo URI part
     *
     * @return string
     */
    public function getPassword();

    /**
     * Set the username part (before the ':') of the userInfo URI part
     *
     * @param  string $user
     * @return \Zend\Uri\Http
     */
    public function setUser($user);

    /**
     * Set the password part (after the ':') of the userInfo URI part
     *
     * @param  string $password
     * @return \Zend\Uri\Http
     */
    public function setPassword($password);

    /**
```

```
 * Validate the host part of an HTTP URI
 *
 * Unlike the generic URI syntax, HTTP URIs do not allow IPv6 or reg-name
 * URIs, only IPv4 and DNS compatible host names.
 *
 * @param  string $host
 * @return boolean
```

```
     */
    static public function validateHost($host);
}
```

```
/**
 * @namespace
 */
namespace Zend\Uri;
use \Zend\Uri\Uri;

/**
 * File URI handler
 *
 * The 'file:...' scheme is loosly defined in RFC-1738
 *
 * @uses      \Zend\Uri\Uri
 * @uses      \Zend\Uri\Exception
 * @category  Zend
 * @package   Zend_Uri
 * @copyright Copyright (c) 2005-2010 Zend Technologies USA Inc. (http://www.zend.com)
 * @license   http://framework.zend.com/license/new-bsd     New BSD License
 */
class File extends Uri
{
    static protected $_validSchemes = array('file');

    /**
     * Check if the URI is a valid File URI
     *
     * This applys additional specific validation rules beyond the ones
     * required by the generic URI syntax
     *
     * @return boolean
     * @see    \Zend\Uri\Uri::isValid()
     */
    public function isValid();

    /**
     * User Info part is not used in file URIs
     *
     * @see    \Zend\Uri\Uri::setUserInfo()
     * @throws InvalidUriPartException
     */
    public function setUserInfo($userInfo);

    /**
     * Fragment part is not used in file URIs
     *
     * @see    \Zend\Uri\Uri::setFragment()
     * @throws InvalidUriPartException
     */
    public function setFragment($fragment);

    /**
     * Convert a UNIX file path to a valid file:// URL
     *
     * @param  srting $path
     * @return \Zend\URI\File
     */
    static public function fromUnixPath($path);

    /**
     * Convert a Windows file path to a valid file:// URL
     *
     * @param  string $path
     * @return \Zend\URI\File
     */
    static public function fromWindowsPath($path);
}
```

The ZendUriUriFactory class

```
namespace Zend\Uri;

/**
 * URI Factory Class
 *
 * The URI factory can be used to create URI objects from strings, using a
 * different URI subclass depending on the input URI scheme. New scheme-specific
 * classes can be registered using the registerScheme() method.
 *
 * Note that this class contains only static methods and should not be
 * instantiated
 *
 * @uses      \Zend\Uri\Uri
 * @uses      \Zend\Uri\Http
 * @uses      \Zend\Uri\File
 * @uses      \Zend\Uri\Exception
 * @uses      \Zend\Loader
 */
abstract class UriFactory
{
    /**
     * Registered scheme-specific classes
     *
     * @var array
     */
    static protected $_schemeClasses = array(
        'http'   => '\Zend\Uri\Http',
        'https'  => '\Zend\Uri\Http',
        'file'   => '\Zend\Uri\File'
    );

    /**
     * Register a scheme-specific class to be used
     *
     * @param unknown_type $scheme
     * @param unknown_type $class
     */
    static public function registerScheme($scheme, $class);

    /**
     * Create a URI from a string
     *
     * @param  string $uri
     * @param  string $defaultScheme
     * @return \Zend\Uri\Uri
     */
    static public function factory($uriString, $defaultScheme = null);
}
```

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