Zend_Uri Improvements - 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 Improvements Component Proposal

Proposed Component Name	Zend_Uri Improvements
Developer Notes	http://framework.zend.com/wiki/display/ZFDEV/Zend_Uri Improvements
Proposers	shahar.e@zend.com
Revision	0.1 - 13 October 2007: Initial Proposal (wiki revision: 16)

Table of Contents

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

1. Overview

Zend_Uri is a component designed to represent and ease up the use of Uniform Resource Identifiers (URIs) of various schemes. It's current implementation (it is one of the least-modified and earliest components of ZF) is somewhat lacking - providing very little benefit besides validation (which might already be provided as part of Zend_Validate) and parsing, which can be done using native PHP functions such as parse_url(). Additionally, it does not support the representation of abstract (non scheme specific) or partial URIs.

The aim of this proposal is to describe a set of changes to Zend_Uri that will improve the ability to represent abstract URIs and new scheme-specific URIs, as well as add some required functionality, without minimal API changes, while maintaining as much backwards compatibility as possible.

2. References

- Current Zend_Uri Reference Page
- RFC 3986 - Uniform Resource Identifier (URI): Generic Syntax
- RFC 2616 (HTTP) Section on HTTP URIs
- RFC 2368 - The mailto URL scheme
- "URI Clarification" at the W3C

3. Component Requirements, Constraints, and Acceptance Criteria

- This component **will** allow the programmatic construction of URIs from scratch, as well as construction by string parsing
- This component **will** allow reading and modifying each part of the URI, except for the scheme in scheme-specific subclasses
- This component **will** provide part or URI validation where applicable, relying on Zend_Validate where possible
- This component **will** allow the retrieval of the full URI as a string
- This component **will** allow the representation of abstract generic URIs
- This component **will** be fully extensible to allow specific URI schemes
    - The following scheme-specific URI classes will be implemented as part of the component:
        - Zend_Uri_Http - HTTP / HTTPS scheme
        - Zend_Uri_Mailto - Mailto scheme
        - Zend_Uri_File - File scheme (optional)
    - This component **will** allow the easy use of user-defined classes for other URI schemes
- This component **will** allow the representation of partial or relative URIs
- This component **will** provide a method to use an absolute URL as the base URL for a relative or partial URI, translating it into an absolute full URL.
- This component **will** allow automatic instantiation of different URI subclasses through a factory pattern.
- This component **will** aim to be as lightweight as possible, not adding more logic than required by this proposal, following the assumption that Zend_Uri will mostly be used as an underlying helper class
- This component **will not** provide part-specific validation functions (as the old implementation did) as those seem redundant and are probably rarely used directly by the end-user.
- The changes to this component **will** try to break backwards compatibility with the old implementation only when necessary

4. Dependencies on Other Framework Components

- Zend_Loader
- Zend_Validate
- Zend_Exception

5. Theory of Operation

Instantiation and manipulation

When parsing string representation of URIs into Zend_Uri objects, one will usually either instatiate a specific Zend_Uri_Xxx() object, passing the string to the constructor, or use the generic Zend_Uri::factory() method, which will decide which specific class to instantiate. If no specific class exists for this URI, or the scheme is unknown, the generic Zend_Uri class will be used to represent the scheme. In any case all parsing will be based on the internal parse_url() function, which seem to be good enough for all the current needs.

One could also instantiate an empty Uri object and construct a URI programmatically, either by manually setting each one of the parts or by passing an array (similar in structure to the return value of parse_url()) to the constructor of the class.

When manually constructing, classes should validate that the parts are well formed. Additionally, when stringifying a URI object, required parts must exist (for example if the scheme is 'http' but there is no host, an exception should be thrown).

The generic factory method will try to match the scheme (if available) to one of it's subclasses. A static method must be provided to allow the user to define additional classes for specific schemes - these classes must extend Zend_Uri. Once a class is matched to the scheme, a new object of this class will be returned. If no class is matched, or if the URI has no scheme (as is the case in relative URIs) an instance of Zend_Uri will be returned.

Subclassing

The basic Zend_Uri will no longer be abstract (as it is now) but will be a real class, with minimal URI representation (scheme, user, host, port) and manipulation capabilities for generic purpose. In order to provide scheme-specific functionality, this class will be extended by several subclasses - for example Zend_Uri_Http will allow HTTP specific functionality such as manipulation of the HTTP query string or fragment representation.

In addition to the provided subclasses, the user will also be able to extend Zend_Uri in order to represent other specific schemes, and will be able to register his specific classes in Zend_Uri so that they could be used by it's factory pattern.

Merging and Reference Resolution

One of the biggest additions to be added on top of the old implementation will be the ability to merge two partial URIs or an absolute and relivate ones into a single new URI, using one URI as the base URI for the second one. This will be done in accordance to the 5. Reference Resolution chapter in the URI RFC. Currently, two methods are proposed to handle this functionality:

1. Passing a base URI object to the URI constructor in addition to the new URI string (or array), allowing the constructor to use this base URI if needed when creating the new URI.
2. Through a static merge() method which will clone and return a new URI object merged from two passed URI objects.

6. Milestones / Tasks

- Milestone 1: design notes will be published here
- Milestone 2: Working prototype passing all unit tests of previous versions checked into the incubator
- Milestone 3: Additional unit tests added for new functionality
- Milestone 4: New functionality (merging, subclassing) committed to repository
- Milestone 5: Initial documentation exists.

7. Class Index

- Zend_Uri
- Zend_Uri_Http
- Zend_Uri_Mailto
- Zend_Uri_File
- Zend_Uri_Exception

8. Use Cases

UC-1: Instantiating

```
// From a string
$uri = new Zend_Uri_Http('http://user:p4s5w0r3@www.example.com:8000/foo/bar?param=value#fragment');

// From a string using the factory method
$uri = Zend_Uri::factory('http://user:p4s5w0r3@www.example.com:8000/foo/bar?param=value#fragment');

// Different scheme
$uri = Zend_Uri::factory('myScheme://somehost:1000/path');

// Empty mailto URI object
$uri = new Zend_Uri_Mailto();

// Mailto URI from array
$uri = new Zend_Uri_Mailto(array(
    'scheme' => 'mailto',
    'path'   => 'shahar.e@zend.com',
    'query'  => 'subject=something'));
```

UC-2: Setting and getting URI parts

```
$uri = new Zend_Uri_Http('http://www.example.com');
echo $uri->getHost(); // Should print out 'www.example.com'

$uri->setPath('/foo/file.php');
$uri->setQuery(array(
    'param1' => 'value1',
    'param2' => 'value2'
));
$uri->setScheme('https');

echo echo 'New URL: ' . $uri; // __toString() magic converts back to string
// Output should be 'https://www.example.com/foo/file.php?param1=value1&param2=value2'
```

UC-3: Part validation

```
$uri = new Zend_Uri_Mailto();
$uri->setPath('shahar.e@zend.com'); // Will work fine
$uri->setPath('wr@ng guy@example.com') // Will throw an exception because path is not a valid email
address

$uri = new Zend_Uri_Http();
$uri->setHost('www.example.com'); // Will work
$uri->setHost('random stuff'); // Will not work, and throw an exception
```

UC-4: Merging URIs using a base URL

```
$baseUri = Zend_Uri::factory('http://www.example.com/myfile');

// ...Fetch page using Zend_Http_Client and read links from it
// $links is an array of <a href=""> link targets from the body

// Print a list of HTTP links found in the page
foreach ($links as $link) {
    // Build an absolute URI using partial relative links found in page and the
    // base URL of the page. This should work well even if some of the links are
    // absolute (external) full URLs or even mailto: links (basically $baseUri should
    // be ignored if it is not applicable)
    $linkUri = Zend_Uri::factory($link, $baseUri);
    if ($linkUri instanceof Zend_Uri_Http) echo $linkUri;
}
```

UC-5: Merging URIs using the static merge method

```
// Merge a full HTTP URL object with a partial URI object
$uri1 = Zend_Uri::factory('http://www.example.com/foo/bar/file');
$uri2 = Zend_Uri::factory('../baz/something');
$uri3 = Zend_Uri::merge($uri1, $uri2);  // $uri3 is a new object
echo $uri3->getUri(); // Same as __toString() - prints 'http://www.example.com/foo/baz/something'

// Merge a full HTTP URL with a query string. ::merge() works with strings
// as well as with objects - but will always return a Zend_Uri object
$uri1 = 'http://www.example.com/foo/bar';
$uri2 = '?param=value&morestuff=things';
$uri3 = Zend_Uri::merge($uri1, $uri2);
echo $uri3->getUri(); // Same as __toString() - prints
'http://www.example.com/foo/bar?param=value&morestuff=things'

// Same, but with a Mailto URL
$uri1 = new Zend_Uri_Mailto();
$uri1->setPath('someone@example.com');
$mailtoWithSubject = Zend_Uri::merge($uri1, '?subject=SomeSpam');
// $mailtoWithSubject is now an object containing 'mailto:someone@example.com?subject=SomeSpam'

// Will throw an exception, because URIs are not of the same scheme
// This is different from passing a BaseURL to the constructor, which simply silently fails
$uri = Zend_Uri::merge('http://example.com', 'mailto:someone@example.com');
```

UC-6: Support for abstract and user defined URI schemes

```
// Any URI like string can be put into Zend_Uri
$uriObj = Zend_Uri::factory('myshceme://host/path');
$uriObj = Zend_Uri::factory('urn:foo.bar.thing');

// You can also register custom URI subclasses for different schemes
Zend_Uri::addSchemeClass('myscheme', 'Myproj_Uri_Myscheme');
$uriObj = Zend_Uri::factory('myshceme://host/path');
// $uriObj is now an instance of Myproj_Uri_Myscheme which extends Zend_Uri

// The second parameter of ::addSchemeClass is a class name which has to be loadable
// using Zend_Loader::loadClass() - I.E Myproj_Uri_Myscheme has to be defined in
// Myproj/Uri/Myscheme.php under the include_path

// You can also register a set of schemes
Zend_Uri::addSchemeClass(array(
    'myscheme' => 'Myproj_Uri_Myscheme',
    'urn'      => 'MyProj_Uri_Urn',
    'smb'      => 'Someframework_Uri_Smb'
));
```

9. Class Skeletons

```
class Zend_Uri_Exception extends Zend_Exception {}
```

```
class Zend_Uri
{
    /**
     * Scheme of this URI (http, ftp, etc.)
     *
     * @var string
     */
    protected $scheme = null;
```

```
    /**
     * URI hostname
     *
     * @var string
     */
    protected $host   = null;

    /**
     * URI port
     *
     * @var integer
     */
    protected $port   = null;

    /**
     * URI path
     *
     * @var string
     */
    protected $path   = null;

    /**
     * Create a new Zend_Uri object from string or array.
     *
     * @param string|array          $uri
     * @param Zend_Uri|string|array $baseUrl
     */
    public function __construct($uri, $baseUrl = null);

    /**
     * Return a string representation of this URI.
     *
     * @see     getUri()
     * @return  string
     */
    public function __toString();

    /**
     * Load the parsed URI array into the object properties. This method
     * should be different for each Zend_Uri subclass.
     *
     * @param array $uri
     */
    protected function loadParsedUri(array $uri);

    /**
     * Tell whether this is a complete full URL or just a partial or
     * relative one
     *
     * @return boolean
     */
    public function isComplete();

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

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

    /**
     * Get the URI port
     *
```

```
     * @return integer
     */
    public function getPort();

    /**
     * Get the URI's scheme
     *
     * @return string|null Scheme or null if no scheme is set.
     */
    public function getScheme();

    /**
     * Set the URI host
     *
     * @param  string $host
     * @return Zend_Uri
     */
    public function setHost($host);

    /**
     * Set the URI path
     *
     * @param  string $path
     * @return Zend_Uri
     */
    public function setPath($path);

    /**
     * Set the URI port
     *
     * @param  integer $port
     * @return Zend_Uri
     */
    public function setPort($port);

    /**
     * Set the URI scheme
     *
     * @param  string $scheme
     * @return Zend_Uri
     */
    public function setScheme($scheme);

    /**
     * Return a string representation of this URI.
     *
     * @return string
     */
    public function getUri();

    /**
     * Returns TRUE if this URI is valid, or FALSE otherwise.
     *
     * @return boolean
     */
    public function valid();

    /**
     * Convenience function, checks that a $uri string is well-formed
     * by validating it but not returning an object.  Returns TRUE if
     * $uri is a well-formed URI, or FALSE otherwise.
     *
     * @param string $uri
     * @return boolean
     */
    public static function check($uri);

    /**
     * Merge a relative URI to an absolute base URL
     *
```

```
     * @param Zend_Uri|srting $uri
     * @param Zend_Uri|string $baseUrl
     * @return Zend_Uri
     */
    public static function merge($uri, $baseUrl);

    /**
     * Create a new Zend_Uri object from an arbitrary URI. If applicable, will
     * use $baseUrl as the basis for $uri if it is relative.
     *
     * @param  string $uri
     * @param  Zend_Uri|string Base URL
     * @throws Zend_Uri_Exception
     * @return Zend_Uri
```

```
     */
    public static function factory($uri, $baseUrl = null);
}
```

```
class Zend_Uri_Http
{
    /**
     * URI user name
     *
     * @var string
     */
    protected $username = null;

    /**
     * URI password
     *
     * @var string
     */
    protected $password = null;

    /**
     * URI query string
     *
     * @var string
     */
    protected $query    = null;

    /**
     * URI fragment
     *
     * @var string
     */
    protected $fragment = null;

    /**
     * Load the parsed URI array into the object properties. This method
     * should be different for each Zend_Uri subclass.
     *
     * @param array $uri
     */
    protected function loadParsedUri(array $uri);

    /**
     * Tell whether this is a complete full URL or just a partial or
     * relative one
     *
     * @return boolean
     */
    public function isComplete();

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

    /**
     * Get the URI password
     *
     * @return string
     */
    public function getPassword ();

    /**
     * Get the URI port. If no port is set, will return the default
```

```
     * port according to scheme (80 or 443)
     *
     * @return integer
     */
    public function getPort();

    /**
     * Get the URI query
     *
     * @param  boolean $asString Set to FALSE to return query as array
     * @return string|array
     */
    public function getQuery($asString = true);

    /**
     * Get the URI User
     *
     * @return string
     */
    public function getUsername();

    /**
     * Set the fragment part (the part after the '#')
     *
     * @param  string $fragment
     * @return Zend_Uri_Http
     */
    public function setFragment($fragment);

    /**
     * Set the URI password
     *
     * @param  string $password
     * @return Zend_Uri_Http
     */
    public function setPassword($password);

    /**
     * Set the URI query string (the part after the '?')
     *
     * Can be either an array or a string. If it is an array, it will be
     * used as input for the internal http_build_query() function
     *
     * @param  array|string $query
     * @return Zend_Uri_Http
     */
    public function setQuery($query);

    /**
     * Set the URI user name
     *
     * @param  string $user
     * @return Zend_Uri_Http
     */
    public function setUsername($user);

    /**
     * Set the URI scheme. Only allows 'http' or 'https'.
     *
     * @param  string $scheme
     * @return Zend_Uri_Http
     */
    public function setScheme($scheme);

    /**
     * Return a string representation of this URI.
     *
     * @return string
     */
    public function getUri();
```

```
/**
 * Returns TRUE if this URI is valid, or FALSE otherwise.
 *
 * @return boolean
 */
public function valid();

/**
 * Convenience function, checks that a $uri string is well-formed
 * by validating it but not returning an object.  Returns TRUE if
 * $uri is a well-formed URI, or FALSE otherwise.
 *
 * @param string $uri
 * @return boolean
```

```
     */
    public static function check($uri);
}
```

```
class Zend_Uri_Mailto
{
    /**
     * URI query string
     *
     * @var string
     */
    protected $query    = null;

    /**
     * Load the parsed URI array into the object properties. This method
     * should be different for each Zend_Uri subclass.
     *
     * @param array $uri
     */
    protected function loadParsedUri(array $uri);

    /**
     * Tell whether this is a complete full URL or just a partial or
     * relative one
     *
     * @return boolean
     */
    public function isComplete();

    /**
     * Get the URI query. mailto: URIs sometimes have query parts that define
     * mail subject, 'CC' fields, etc.
     *
     * @param  boolean $asString Set to FALSE to return query as array
     * @return string|array
     */
    public function getQuery($asString = true);

    /**
     * Set the URI query string (the part after the '?')
     *
     * Can be either an array or a string. If it is an array, it will be
     * used as input for the internal http_build_query() function
     *
     * @param  array|string $query
     * @return Zend_Uri_Mailto
     */
    public function setQuery($query);

    /**
     * Set the URI scheme. Only allows 'mailto'.
     *
     * @param  string $scheme
     * @return Zend_Uri_Mailto
     */
    public function setScheme($scheme);

    /**
     * Return a string representation of this URI.
     *
     * @return string
     */
    public function getUri();

    /**
     * Returns TRUE if this URI is valid, or FALSE otherwise.
```

```
     *
     * @return boolean
     */
    public function valid();

    /**
     * Convenience function, checks that a $uri string is well-formed
     * by validating it but not returning an object.  Returns TRUE if
     * $uri is a well-formed URI, or FALSE otherwise.
     *
     * @param string $uri
     * @return boolean
```

```
     */
    public static function check($uri);
}
```

```
class Zend_Uri_File
{
    /**
     * Load the parsed URI array into the object properties. This method
     * should be different for each Zend_Uri subclass.
     *
     * @param array $uri
     */
    protected function loadParsedUri(array $uri);

    /**
     * Tell whether this is a complete full URL or just a partial or
     * relative one
     *
     * @return boolean
     */
    public function isComplete();

    /**
     * Set the URI scheme. Only allows 'file'.
     *
     * @param  string $scheme
     * @return Zend_Uri_File
     */
    public function setScheme($scheme);

    /**
     * Return a string representation of this URI.
     *
     * @return string
     */
    public function getUri();

    /**
     * Returns TRUE if this URI is valid, or FALSE otherwise.
     *
     * @return boolean
     */
    public function valid();

    /**
     * Convenience function, checks that a $uri string is well-formed
     * by validating it but not returning an object.  Returns TRUE if
     * $uri is a well-formed URI, or FALSE otherwise.
     *
     * @param string $uri
     * @return boolean
     */
    public static function check($uri);
}
```

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