

Zend_Http_Client 2.0 - Shahar Evron

<ac:macro ac:name="info"><ac:parameter ac:name="title">New Proposal Template</ac:parameter><ac:rich-text-body>
<p>This page has been created from a template that uses "zones." To proceed:</p>

Edit the page
Replace sample content within each zone-data tag with your own content
Remove this notice
Save the page
When you are ready for community review, move this page to the <ac:link><ri:page ri:content-title="Ready for Review" /></ac:link> section on the edit page.

</ac:rich-text-body></ac:macro>

<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_Http_Client 2.0 Component Proposal

Proposed Component Name	Zend_Http_Client 2.0
Developer Notes	http://framework.zend.com/wiki/display/ZFDEV/Zend_Http_Client 2.0
Proposers	Shahar Evron
Zend Liaison	TBD
Revision	0.1 - 26 February 2010: Initial Draft. (wiki revision: 10)

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

This proposal describes the suggested rewrite of Zend_Http_Client and its related classes rewrite for ZF 2.0.

The current implementation of the Zend_Http family of classes was originally developed in the days of Zend Framework 0.2. Since then, a lot of knowledge has been gained in regards to what is required from a flexible, easy to use HTTP infrastructure. Most of the Service components did not exist when Zend_Http_Client was written. Additionally, over time support for different features was added, and the need to maintain backwards compatibility dictated that these features will be added in a very specific manner, sometimes limited in terms of modularity and flexibility. Over the years some bugs and feature requests could not have been fixed, due to the need to maintain backwards compatibility.

The goal of this proposal is to describe a complete reimplemention of the Zend_Http class family and specifically of Zend_Http_Client, based on gained knowledge and feedback from the last 3 years. This implementation will not maintain backwards compatibility with Zend Framework 1.0 HTTP modules.

2. References

- [Zend Framework 1.0 HTTP Client](#)
- [Current bugs in ZF 1.0 HTTP modules](#)
- [RFC 2616 - HTTP/1.1 RFC](#)
- [RFC 2617 - Basic / Digest HTTP Authentication](#)
- [RFC 2965 - HTTP Cookies](#) - compatible with the Netscape Cookie specs which are the de-facto standard
- [RFC 3986 - Generic URI Syntax](#)

3. Component Requirements, Constraints, and Acceptance Criteria

- This component **will** implement an RFC-compliant (or as close as possible to compliant) HTTP client
- This component **may** adjust itself to support popular but non strict HTTP implementations
- This component **will** provide all the features provided by Zend Framework 1.0 HTTP client
- This component **will** decouple the HTTP Request representation from the HTTP Client representation
- This component **will** support advanced HTTP client features beyond the request scope such as redirection handling, proxy support, authentication support, file uploads, cookie persistence etc. through the HTTP Client class
- This component **will** allow sending simple HTTP requests without usage of the more advanced HTTP Client class through direct usage of HTTP Request and HTTP Transport objects
- This component **will** provide easy support for piping HTTP request and response bodies from/to PHP streams in addition to usage of in-memory strings
- This component **will** enforce strict validation of SSL certificates for security reasons by default, and will allow disabling this enforcement
- This component **will not** encode cookies through URL encoding unless explicitly configured to
- This component **will** enable sending multiple HTTP requests through the same open connection in a clean and efficient manner, to provide for keep-alive connections, HTTPS proxying, support for "100 Continue" responses etc.
- This component **will** provide different mechanisms for the underlying HTTP communication transport including native sockets, cURL and a mock transport layer for testing purposes
- This component **may** provide a transport layer based on the pecl_http extension
- This component **will** keep the decoding of HTTP transfer-encoded content in the transport layer and outside of the Response object scope
- This component **may** enable better interoperability with the Zend_Controller HTTP request and response objects
- This component **will** follow the ZF 2.0 coding standards and guidelines, including usage of PHP 5.3 namespaces, acceptance of a Zend_Config object in constructors, and reliance on automatic class loading
- This component **will** be tested against common HTTP servers, including Apache 2.x, Microsoft IIS 6.x and 7.x, and others
- This component **will** isolate all tests that

4. Dependencies on Other Framework Components

- `Zend_Uri_Http`
- `Zend_Config`

5. Theory of Operation

The ZF 2.0 HTTP component family will be organized in the following structure:

Messages: Request and Response objects

`Zend\Http\Message` will be an abstract class defining the shared properties of HTTP messages (requests and responses) including their headers, body, etc. It will have two concrete implementations:

- `Zend\Http\Request` which will represent a single, self-contained HTTP request
- `Zend\Http\Response` which will represent a single, self-contained HTTP response

Message objects will contain as little as logic as possible beyond the representation of requests and responses, and the creation or conversion of them. They will be completely decoupled from the HTTP transport layer and from any workflows involving more than one HTTP request - for example automatically handling HTTP redirections.

Message objects must be able to utilize PHP streams to represent the message body in addition to working with in-memory strings. That is, one could set a PHP stream open for reading to represent the request body, and set a PHP stream open for writing to contain the HTTP response body. This is required in order to facilitate submitting and receiving very large message bodies without overflowing the PHP memory limit.

In contrast to the ZF 1.0 implementation, simple HTTP requests could be sent by simply instantiating a request object and sending it through a transport object, without creating the more complex and feature rich Client object.

The Transport Layer

Sending and receiving HTTP messages to and from HTTP servers will be handled through dedicated Transport objects, in a much similar fashion to how ZF 1.0 `Zend\Http\Client\Adapter` classes work. All Transport classes will inherit from single abstract class or interface, and will implement the ability to manage connections and to send request messages and receive response messages. Different concrete **Zend\Http\Transport** implementation could be based on different underlying infrastructure such as native sockets, `cURL`, `pecl_http` etc.

A few notable differences from today's Adapter design:

- Transport objects will communicate through HTTP message objects (request and response) and not through strings and arrays
- Transport objects will encapsulate and "hide" the logic of decoding transfer-encoded HTTP response bodies (today this is partially handled by response objects)
- Transport objects will not attempt to automatically handle different protocol negotiation issues that require more than one request to be sent or acting upon an unexpected or special response - for example, sending two requests in order to facilitate the HTTP CONNECT method for HTTPS proxying. This will be left to Client objects.

The Client class

Client objects will encapsulate all the more complex HTTP communication logic which is beyond the scope of simple, single HTTP request and response. For example, the following functionality will be handled by the HTTP Client class:

- Automatic redirection response handling
- Authentication handling through a convenient API (without the need to manually set request headers)
- Cookie persistence (replacing the existing `Zend\Http\CookieJar` class)
- HTTP Proxy support
- Handling file uploads

The Client class will also provide some convenience methods to quickly send HTTP requests of common types (GET, POST, PUT, DELETE).

Auxiliary Classes

Additional auxiliary classes could be implemented if needed. The following classes are considered:

- `Zend\Http\Cookie` - representation an HTTP cookie
- `Zend\Http\Message\Body` - representation of an HTTP message body, might be needed in order to facilitate streams vs. in-memory transparency
- `Zend\Http\Request\FileUpload` - a more complex Request object providing the ability to send a POST request containing file uploads (i.e. way sent by browsers when using the `<input type="file">` form field) where files are read directly from streams

6. Milestones / Tasks

- Milestone 1: Design notes are published here
- Milestone 2: Design notes are reviewed, discussed and generally agreed upon by the ZF community
- Milestone 3: A working prototype is checked into the incubator and is made available for initial testing
- Milestone 4: Design refinement and final acceptance by the ZF community and core development team

- Milestone 5: A final implementation is checked into the ZF incubator, possibly depending on the availability of ZF 2.0 ports of Zend_Uri and Zend_Config
- Milestone 6: 90%+ unit test coverage, documentation is written
- Milestone 7: Final code is committed to the ZF 2.0 release branch

7. Class Index

- Zend\Http\Client
- Zend\Http\Message\MessageAbstract
- Zend\Http\Request
- Zend\Http\Response
- Zend\Http\Transport\TransportAbstract
- Zend\Http\Transport\Socket
- Zend\Http\Transport\Curl
- Zend\Http\Transport\PecHttp
- Zend\Http\Transport\Dummy
- Zend\Http\Cookie

8. Use Cases

UC-01

Sending a simple GET request quickly and checking the response status:

```
$response = \Zend\Http\Client::get('http://example.com/foo/bar');
if ($response->isSuccess()) {
    // do something
}
```

Here's another method of sending a GET request quickly:

```
$response = \Zend\Http\Transport\Socket::send(new
\Zend\Http\Request('http://example.com/foo/bar'));
```

UC-02

Sending a POST request with some URL-encoded parameters

```
$params = array(
    'war'     => 'peace',
    'freedom' => 'slavery',
    'banana' => 'apple'
);

$response = \Zend\Http\Client::post('http://example.com/', $params);
```

Another method of doing the same thing, this time with more control over the request:

```
$request = new \Zend\Http\Request('http://example.com/', 'POST');
$request->setBody(http_build_query($params), 'application/x-form-urlencoded');
$response = $client->send($request);
```

Adding additional request headers

```
$request = new \Zend\Http\Request('http://example.com/', 'POST');
$request->setHeader('From', 'nobody@example.com')
    ->setHeaders(array(
        'Accept'           => 'text/xml, text/html, */*',
        'Accept-charset'   => 'utf8, *;q=0.8',
        'Accept-encoding' => 'identity'
    ));

$response = $client->send($request);
```

9. Class Skeletons

```
namespace Zend\Http\Message;

abstract class MessageAbstract
{
    /**
     * HTTP version constants
     */
    const HTTP_VER_10 = '1.0';
    const HTTP_VER_11 = '1.1';

    /**
     * Message headers
     *
     * @var array
     */
    protected $_headers = array();

    /**
     * HTTP version
     *
     * @var string
     */
    protected $_httpVersion = self::HTTP_VER_11;

    /**
     * Set a header
     *
     * @param string $name
     * @param string $value
     * @return \Zend\Http\Message\MessageAbstract
     */
    public function setHeader($name, $value)
    { }

    /**
     * Set several headers by passing an associative array
     */
}
```

```

*
* @param array $headers
* @return \Zend\Http\Message\MessageAbstract
*/
public function setHeaders(array $headers)
{ }

/**
 * Get the value of the header $name, or null if it does not exist
 *
 * @param string $name
 * @return string | null
 */
public function getHeader($name)
{ }

/**
 * Get all headers as an array
 *
 * @return array
 */
public function getHeaders()
{ }

/**
 * Get all headers as a string
 *
 * @param $eol Optional End Of Line string
 * @return string
 */
public function getHeadersAsString($eol = "\r\n")
{ }

/**
 * Set the message's HTTP version
 *
 * @param string $ver
 * @return \Zend\Http\Message\MessageAbstract
 */
public function setHttpVersion($ver)
{ }

/**
 * Get the message's HTTP version
 *
 * @return string
 */
public function getHttpVersion()
{ }

/**
 * Convert the message to a string
 *
 * @return string
 */
public function __toString()
{
    return $this->_getStartLine() . "\r\n" .
        $this->getHeadersAsString() . "\r\n" .

```

```
        $this->getBody();
    }

    /**
     * Get the message body
     *
     * @return string
     */
    abstract public function getBody();

    /**
     * Get the message start line - this is message type dependant
     *
     * @return string
     */
    abstract protected function _getStartLine();

    /**
     * Static helper methods
     */

    /**
     * Check if the provided string is a valid token
     *
     * @link http://www.w3.org/Protocols/rfc2616/rfc2616-sec2.html#sec2.2
     * @param string $str
     * @return boolean
     */
    static protected function _isToken($str)
```

```
{ }  
}
```

```
namespace Zend\Http;  
use \Zend\Http\Message\MessageAbstract as MessageAbstract;  
  
class Request extends MessageAbstract  
{  
    const GET      = 'GET';  
    const POST     = 'POST';  
    const PUT      = 'PUT';  
    const DELETE   = 'DELETE';  
    const HEAD     = 'HEAD';  
    const OPTIONS  = 'OPTIONS';  
    const TRACE    = 'TRACE';  
    const CONNECT  = 'CONNECT';  
  
    protected $_method;  
  
    protected $_uri;  
  
    protected $_body;  
  
    public function __construct($uri, $method = 'GET', array $headers = array(), $body  
= null)  
    { }  
  
    public function setUri($uri)  
    { }  
  
    public function setMethod($method)  
    { }  
  
    public function setBody($body)  
    { }  
  
    public function getBody()  
    { }  
  
    protected function _getStartLine()  
    { }  
}
```

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