Pingback

The Pingback class is a pingback 1.0 protocol (http://www.hixie.ch/specs/pingback/pingback) (client and server) implementation.

Namespace: `Web`
File location: `lib/web/pingback.php`

Requirements

The Pingback class needs the PHP XML-RPC extension (http://php.net/manual/en/book.xmlrpc.php). The Pingback class won't work without it.

You can easily check for its availability on your server this way:

```
$isPingbackAvailable = extension_loaded('xmlrpc');  // returns TRUE or FALSE
```

Instantiation

Return class instance

```
$pingback = Web\Pingback::instance();
```

The Pingback class uses the Prefab (prefab-registry) factory wrapper, so you can grab the same instance of that class at any point of your code.

Methods

inspect

Load local page contents, parse HTML anchor tags, find permalinks, and send XML-RPC calls to corresponding pingback servers

```
NULL inspect ( string $source )
```

This function performs a web request to load the local page contents given by the `$source` URL. Then the retrieved response is parsed, looking for HTML anchor tags. For every permalink found (`<link rel="pingback" />` typically) inside the page, a XML-RPC call is send to the corresponding pingback server (`METHOD POST`). (Every XML-RPC call is logged. See function `log()` below to retrieve transactions history.)

Example:

```
$pingback->inspect($source);
```

listen

Receive ping, check if local page is pingback-enabled, verify source contents, and return XML-RPC response

```
string listen ( callback $func [ , string $path = NULL ] )
```

This function allows you to setup a XML-RPC listener. You need to define a F3 route and bind it to this function.

The function basically 'listen', once bound to a F3 route, to receive a ping, checks then if the local page is pingback-enabled, verifies the source contents, and for each link found in the source contents, use a given `$func` callback function to return a XML-RPC response.

The `$func` parameter is the name of your callback function to use to handle the request/ping. It will be called like this:

```
call_user_func_array($func, array($source,$req['body']));
```

with `$source` being the URL of the source, and `$req['body']` the body part returned

If `$path` is not provided, the value of the BASE (quick-reference#base) system variable is used.

On success, this function will `die` returning the XML response content as per the xmlrpc_encode_request (http://php.net/manual/en/function.xmlrpc-encode-request.php) php function:

```
// Success
die(xmlrpc_encode_request(NULL,$source,$options));
```

On error, this function will `die` returning the XML response consisting of one of the following error code as follow:

```
die(xmlrpc_encode_request(NULL,0x11,$options));  // No link to local page found in re
quest body

die(xmlrpc_encode_request(NULL,0x10,$options));  // Source failure: web request faile
d or received doc malformed

die(xmlrpc_encode_request(NULL,0x21,$options));  // Local page doesn't exist or is no
t pingback-enabled

die(xmlrpc_encode_request(NULL,0x31,$options));  // Access denied: request method is
not 'pingback.ping' or request malformed
```

Now, let's setup a basic example of a XML-RPC ping handler:

We need a route, a listening function to bind to, and a callback function:

```
// the callback function called by the listen() function. Takes 2 parameters
function pingCallBackHandler($sourceURL, $reqBody) {
    $logger = new Log('pings.log');
    $logger->write('Incoming ping from '.$sourceURL);
    // any processing on the $reqBody
    $logger->write('Request body length is '.
            \UTF::instance()->strlen($reqBody));
}

// a route as an example, e.g.
$f3->route('GET /listener','PingListener');

// the function to bind to the listener:
function PingListener($f3, $params) {

    $pingback = new \Web\Pingback;
    // bind it with our custom callback function
    $pingback->listen('pingCallBackHandler');
    return;
}
```

log

Return transaction history

```
string log ( )
```

This function returns the transaction history as logged by the `inspect()` function. The transaction history consists of a list of the permalinks URLs found in every inspected page, and this for every request response.

Example:

```
echo $pingback->log();


// Outputs:
Mon, 06 Jan 2014 10:23:00 +0100 /comments-feed?page=pingback/cf [permalink:/pingback]
   /pingback2?page=pingback/client

Mon, 06 Jan 2014 10:23:01 +0100 /rss2-feed?page=pingback/rss2 [permalink:/rss2-ping]
 /pingback2?page=rss2-ping/client
```

__construct

Instantiate class

```
object __construct ( )
```

The constructor allows you to instantiate the class.

Example:

```
$pingback = new Pingback (  )
```

Notice: As a convenience, this constructor calls libxml_use_internal_errors(TRUE) (http://php.net/manual/en/function.libxml-use-internal-errors.php) to suppress errors caused by invalid HTML structures.

enabled

Return TRUE if URL points to a pingback-enabled resource

```
protected bool enabled ( $url )
```

This function returns TRUE if the given URL points to a pingback-enabled resource.

This is a *protected* function used internally by `inspect()` and `listen()` to make sure a given URL points to a pingback-enabled resource, i.e. it looks for a valid pingback header and scan the page to make sure it contains pingback link tag(s).