HTTP

Introduction

This HTTP extension aims to provide a convenient and powerful set of functionality for one of PHPs major applications.

It eases handling of HTTP URLs, dates, redirects, headers and messages, provides means for negotiation of clients preferred language and charset, as well as a convenient way to send any arbitrary data with caching and resuming capabilities.

It provides powerful request functionality, if built with CURL support. Parallel requests are available for PHP 5 and greater.

Additionally to the API reference in this manual you can find information about how to install and configure as well as which global constants are predefined in the following sections:

Installing/Configuring

Requirements

The HttpResponse class requires at least PHP v5.1. Any other class is available as of PHP v5.0.

Note
Be aware though, that some methods are not available with PHP v5.0.

Installation requirements on Windows

In order to be able to load this extension on *Windows*, you additionally need to load the following PHP extensions: hash, iconv and SPL.

Installation requirements on other platforms

The extension must be built with » libcurl support to enable *request* functionality (*--with-http-curl-requests*). A library version equal or greater to *v7.12.3* is required.

To enable support for sending and receiving *compressed* responses, the extension must be built with » zlib support (*--with-http-zlib-compression*). A library version equal or greater than *v1.2.2* is required.

Content type quessing can be enabled by building this extension with » libmagic support (*--with-http-magic-mime*).

Installation/Configuration

Installation

This » PECL extension is not bundled with PHP.

Information for installing this PECL extension may be found in the manual chapter titled Installation of PECL extensions. Additional information such as new releases, downloads, source files, maintainer information, and a CHANGELOG, can be located here: » http://pecl.php.net/package/pecl_http.

Note
The official name of this extension is *pecl_http*.

The DLL for this PECL extension may be downloaded from either the » PHP Downloads page or from » http://pecl4win.php.net/

Runtime Configuration

The behaviour of these functions is affected by settings in *php.ini*.

HTTP Configuration Options

Name	Default	Changeable	Description
http.etag.mode	"MD5"	PHP_INI_ALL	The hashing algorithm used to generate the ETag. MD5, SHA1, and CRC32 are always available. If the hash xtension is enabled, any hashing algorithms this extension provides are available, too.
http.log.cache	""	PHP_INI_ALL	The path (or stream wrapper url) to a log file in which to write successful cache hits.
http.log.redirect	""	PHP_INI_ALL	The path (or stream wrapper url) to a log file in which to write redirects.
http.log.not_found	""	PHP_INI_ALL	The path (or stream wrapper url) to a log file in which to write "file not found" errors.
http.log.allowed_methods	""	PHP_INI_ALL	The path (or stream wrapper url) to a log file in which to write

			"allowed methods" violations.
http.log.composite	""	PHP_INI_ALL	The path (or stream wrapper url) to a log file in which to write all events.
http.request.methods.allowed	""	PHP_INI_ALL	Allowed request methods. If a client issues a request with a request method other than listed here, PHP exits with a status of "405 Method not allowed". See the INI setting http.force_exit for what "exits" means.
http.request.methods.custom	""	PHP_INI_PERDIR|PHP_INI_SYSTEM	Custom request methods. If you want to use any non-standard request methods, you can register them with this INI setting or http_request_method_register().
http.request.datashare.cookie	"0"	PHP_INI_SYSTEM	Whether the global HttpRequestDataShare should by default share cookie information.
http.request.datashare.dns	"1"	PHP_INI_SYSTEM	Whether the global HttpRequestDataShare should by default share name lookup information.
http.request.datashare.ssl	"0"	PHP_INI_SYSTEM	Whether the global HttpRequestDataShare should by default share SSL session information. This is not yet implemented in libcurl.
http.request.datashare.connect	"0"	PHP_INI_SYSTEM	Whether the global HttpRequestDataSha

			re should by default share connect information. This is not yet implemented in libcurl.
http.persistent.handles.limit	"-1"	PHP_INI_SYSTEM	The maximum amount of persistent handles to keep alive.
http.persistent.handles.ident	"GLOBAL"	PHP_INI_ALL	The ident of persistent handles.
http.send.inflate.start_auto	"0"	PHP_INI_PERDIR|PHP_INI_SYSTEM	Whether to automatically start the inflate output handler.
http.send.inflate.start_flags	"0"	PHP_INI_ALL	Initialization settings for the inflate output handler.
http.send.deflate.start_auto	"0"	PHP_INI_PERDIR|PHP_INI_SYSTEM	Whether to automatically start the deflate output handler.
http.send.deflate.start_flags	"0"	PHP_INI_ALL	Initialization settings for the deflate output handler. See deflate constants.
http.send.not_found_404	"1"	PHP_INI_ALL	Whether to automatically exit with a status of "404 Not found", if http_send_file() was not able to find the specified file. See the INI setting http.force_exit for what "exits" means.
http.only_exceptions	"0"	PHP_INI_ALL	Whether all notices/warnings/errors should be thrown as exceptions.
http.force_exit	"1"	PHP_INI_ALL	Each occasion where "exits with a status of..." is mentioned, usually causes the

			halt of the scripting engine. Disable this option if you alternatively want to start a discarding (dev/null) output handler and continue script execution.

Resource Types

This extension defines a stream resource returned by http_get_request_body_stream() and (thereafter) HttpResponse::getStream().

Predefined Constants

The constants below are defined by this extension, and will only be available when the extension has either been compiled into PHP or dynamically loaded at runtime.

Constants usable with http_support()

HTTP_SUPPORT (integer)
querying for this constant will always return **TRUE**

HTTP_SUPPORT_REQUESTS (integer)
whether support to issue HTTP requests is given, ie. libcurl support was compiled in

HTTP_SUPPORT_MAGICMIME (integer)
whether support to guess the Content-Type of HTTP messages is given, ie. libmagic support was compiled in

HTTP_SUPPORT_ENCODINGS (integer)
whether support for zlib encodings is given, ie. libz support was compiled in

HTTP_SUPPORT_SSLREQUESTS (integer)
whether support to issue HTTP requests over SSL is givem, ie. linked libcurl was built with SSL support

Constants usable with http_parse_params()

HTTP_PARAMS_ALLOW_COMMA (integer)
allow commans additionally to semicolons as separator

HTTP_PARAMS_ALLOW_FAILURE (integer)
continue parsing after an error occurred

HTTP_PARAMS_RAISE_ERROR (integer)
raise PHP warnings on parse errors

HTTP_PARAMS_DEFAULT (integer)
all three values above, bitwise or'ed

Constants usable with http_parse_cookie() **and its return value**

HTTP_COOKIE_PARSE_RAW (integer)
don't urldecode values

HTTP_COOKIE_SECURE (integer)
whether "secure" was found in the cookie's parameters list

HTTP_COOKIE_HTTPONLY (integer)
whether "httpOnly" was found in the cookie's parameter list

Constants usable with http_deflate() **and HttpDeflateStream**

HTTP_DEFLATE_LEVEL_DEF (integer)

HTTP_DEFLATE_LEVEL_MIN (integer)

HTTP_DEFLATE_LEVEL_MAX (integer)

HTTP_DEFLATE_TYPE_ZLIB (integer)

HTTP_DEFLATE_TYPE_GZIP (integer)

HTTP_DEFLATE_TYPE_RAW (integer)

HTTP_DEFLATE_STRATEGY_DEF (integer)

HTTP_DEFLATE_STRATEGY_FILT (integer)

HTTP_DEFLATE_STRATEGY_HUFF (integer)

HTTP_DEFLATE_STRATEGY_RLE (integer)

HTTP_DEFLATE_STRATEGY_FIXED (integer)

Constants usable with HttpDeflateStream and HttpInflateStream

HTTP_ENCODING_STREAM_FLUSH_NONE (integer)
don't flush

HTTP_ENCODING_STREAM_FLUSH_SYNC (integer)
synchronized flush only

HTTP_ENCODING_STREAM_FLUSH_FULL (integer)
full data flush

Constants used for error reporting and Exceptions

HTTP_E_RUNTIME (integer)
runtime error

HTTP_E_INVALID_PARAM (integer)
an invalid parameter was passed

HTTP_E_HEADER (integer)
header() or similar operation failed

HTTP_E_MALFORMED_HEADERS (integer)
HTTP header parse error

HTTP_E_REQUEST_METHOD (integer)
unknown/invalid request method

HTTP_E_MESSAGE_TYPE (integer)
with operation incompatible message type

HTTP_E_ENCODING (integer)

encoding/decoding error

HTTP_E_REQUEST (integer)
request failure

HTTP_E_REQUEST_POOL (integer)
request pool failure

HTTP_E_SOCKET (integer)
socket exception

HTTP_E_RESPONSE (integer)
response failure

HTTP_E_URL (integer)
invalid URL

HTTP_E_QUERYSTRING (integer)
querystring operation failure

Constants usable with HttpMessage

HTTP_MSG_NONE (integer)
the message is of no specific type

HTTP_MSG_REQUEST (integer)
request style message

HTTP_MSG_RESPONSE (integer)
response style message

Constants usable with HttpQueryString

HTTP_QUERYSTRING_TYPE_BOOL (integer)

HTTP_QUERYSTRING_TYPE_INT (integer)

HTTP_QUERYSTRING_TYPE_FLOAT (integer)

HTTP_QUERYSTRING_TYPE_STRING (integer)

HTTP_QUERYSTRING_TYPE_ARRAY (integer)

HTTP_QUERYSTRING_TYPE_OBJECT (integer)

Constants used for the *httpauthtype* request option

HTTP_AUTH_BASIC (integer)
use "basic" authentication

HTTP_AUTH_DIGEST (integer)
use "digest" authentication

HTTP_AUTH_NTLM (integer)
use "NTLM" authentication

HTTP_AUTH_GSSNEG (integer)
use "GSS-NEGOTIATE" authentication

HTTP_AUTH_ANY (integer)
try any authentication scheme

Constants used for the HTTP *protocol* **version** request option

HTTP_VERSION_ANY (integer)
no specific HTTP protocol version

HTTP_VERSION_1_0 (integer)
HTTP version 1.0

HTTP_VERSION_1_1 (integer)
HTTP version 1.1

Constants used for the SSL *protocol* **type and version** request option

HTTP_SSL_VERSION_ANY (integer)
no specific SSL protocol version

HTTP_SSL_VERSION_TLSv1 (integer)
use TLSv1 only

HTTP_SSL_VERSION_SSLv3 (integer)
use SSLv3 only

HTTP_SSL_VERSION_SSLv2 (integer)
use SSLv2 only

Constants used for the *proxytype* request option

HTTP_PROXY_SOCKS4 (integer)
the proxy is a SOCKS4 type proxy

HTTP_PROXY_SOCKS5 (integer)
the proxy is a SOCKS5 type proxy

HTTP_PROXY_HTTP (integer)
standard HTTP proxy

Constants used for the *ipresolve* request option

HTTP_IPRESOLVE_V4 (integer)
use IPv4 only for name lookups

HTTP_IPRESOLVE_V6 (integer)
use IPv6 only for name lookups

HTTP_IPRESOLVE_ANY (integer)
use any IP mechanism only for name lookups

Predefined HTTP request method constants

HTTP_METH_GET (integer)

HTTP_METH_HEAD (integer)

HTTP_METH_POST (integer)

HTTP_METH_PUT (integer)

HTTP_METH_DELETE (integer)

HTTP_METH_OPTIONS (integer)

HTTP_METH_TRACE (integer)

HTTP_METH_CONNECT (integer)

HTTP_METH_PROPFIND (integer)

HTTP_METH_PROPPATCH (integer)

HTTP_METH_MKCOL (integer)

HTTP_METH_COPY (integer)

HTTP_METH_MOVE (integer)

HTTP_METH_LOCK (integer)

HTTP_METH_UNLOCK (integer)

HTTP_METH_VERSION_CONTROL (integer)

HTTP_METH_REPORT (integer)

HTTP_METH_CHECKOUT (integer)

HTTP_METH_CHECKIN (integer)

HTTP_METH_UNCHECKOUT (integer)

HTTP_METH_MKWORKSPACE (integer)

HTTP_METH_UPDATE (integer)

HTTP_METH_LABEL (integer)

HTTP_METH_MERGE (integer)

HTTP_METH_BASELINE_CONTROL (integer)

HTTP_METH_MKACTIVITY (integer)

HTTP_METH_ACL (integer)

Constants usable with http_redirect()
HTTP_REDIRECT (integer)
guess applicable redirect method

HTTP_REDIRECT_PERM (integer)
permanent redirect (*301 Moved permanently*)

HTTP_REDIRECT_FOUND (integer)
standard redirect (*302 Found*)

Note
RFC 1945 and RFC 2068 specify that the client is not allowed to change the method on the redirected request. However, most existing user agent implementations treat 302 as if it were a 303 response, performing a GET on the Location field-value regardless of the original request method. The status codes 303 and 307 have been added for servers that wish to make unambiguously clear which kind of reaction is expected of the client.

HTTP_REDIRECT_POST (integer)
redirect applicable to POST requests (*303 See other*)

HTTP_REDIRECT_PROXY (integer)
proxy redirect (*305 Use proxy*)

HTTP_REDIRECT_TEMP (integer)
temporary redirect (*307 Temporary Redirect*)
Constants usable with http_build_url()
HTTP_URL_REPLACE (integer)
replace every part of the first URL when there's one of the second URL

HTTP_URL_JOIN_PATH (integer)
join relative paths

HTTP_URL_JOIN_QUERY (integer)
join query strings

HTTP_URL_STRIP_USER (integer)
strip any user authentication information

HTTP_URL_STRIP_PASS (integer)
strip any password authentication information

HTTP_URL_STRIP_AUTH (integer)
strip any authentication information

HTTP_URL_STRIP_PORT (integer)
strip explicit port numbers

HTTP_URL_STRIP_PATH (integer)
strip complete path

HTTP_URL_STRIP_QUERY (integer)
strip query string

HTTP_URL_STRIP_FRAGMENT (integer)
strip any fragments (#identifier)

HTTP_URL_STRIP_ALL (integer)
strip anything but scheme and host

Options usable with the HttpRequest class and request functions

Options related to time outs

timeout (integer)
    seconds the whole request may take to complete

connecttimeout (integer)
    seconds the connect, including name resolving, may take

dns_cache_timeout (integer)
    seconds after an dns cache entry times out

Options related to urls

url (string)
    the request url

port (integer)
    use another port as specified in the url

redirect (integer)
    whether and how many redirects to follow; defaults to 0

unrestrictedauth (bool)
    whether to continue sending credentials on redirects to a different host

referer (string)
    the refererring url to send

Options related to cookies

encodecookies (bool)
    whether custom cookies should be urlencode() d prior sending

cookies (array)
    list of cookies as associative array like *array("cookie" => "value")*

cookiestore (string)
    path to a file where cookies are/will be stored

cookiesession (bool)
    don't load session cookies from cookiestore if **TRUE**

Options related to headers

useragent (string)
    the user agent to send; defaults to *PECL::HTTP/x.y.z (PHP/x.y.z)*; omitted if explicitly set to an empty string

lastmodified (int)
    timestamp for If-(Un)Modified-Since header

etag (string)
    quoted etag for If-(None-)Match header

headers (array)
    list of custom headers as associative array like *array("header" => "value")*

Options related to authentication

httpauth (string)
    http credentials in "user:pass" format

httpauthtype (int)
    HTTP authentication type constant

(array)

Options related to proxies

proxyhost (string)
    proxy host in "host[:port]" format

proxyport (int)
    use another proxy port as specified in proxyhost

proxytype (int)
    HTTP proxy type constant

proxyauth (string)
    proxy credentials in "user:pass" format

proxyauthtype (int)
    HTTP authentication type constant

Options related to the transfer

compress (bool)
    whether to request and accept a gzip/deflate content encoded response

resume (int)
    start the download at the specified byte offset if server support is given (indicated by a 206 response code)

range (array)
    array of arrays, each containing two integer s, specifying the ranges to download if server support is given (indicated by a 206 response code); only recognized if the resume option is empty

Options imposing limits

maxfilesize (integer)
    maximum file size that should be downloaded; has no effect, if the size of the requested entity is unknown (eg. dynamic pages with chunked transfer encoding etc.)

low_speed_limit (int)
    the lowest transfer speed a successful request may have

low_speed_time (int)
    the time in which *low_speed_limit* must be transferred for a successful request

max_send_speed (int)
    maximum send speed in bytes per second

max_recv_speed (int)
    maximum receive speed in bytes per second

Callback options

onprogress (callback)
    progress callback

Network options

interface (string)
    outgoing network interface (ifname, ip or hostname)

portrange (array)
    2 integers specifying outgoing portrange to try

SSL options

ssl (array)

Note
SSL options are set through an array with the single "ssl" request option name.

cert (string)
    path to certificate

certtype (string)
    type of certificate

certpasswd (string)
    password for certificate

key (string)
    path to key

keytype (string)
    type of key

keypasswd (string)
    pasword for key

engine (string)
    ssl engine to use

version (int)
    ssl version to use

verifypeer (bool)
    whether to verify the peer

verifyhost (bool)
    whether to verify the host

cipher_list (string)
    list of allowed ciphers

cainfo (string)

capath (string)

random_file (string)

egdsocket (string)

The HttpDeflateStream class

Class synopsis

HttpDeflateStream

HttpDeflateStream {

public void **HttpDeflateStream::__construct** ([int `$flags = 0`])

public HttpDeflateStream **HttpDeflateStream::factory** ([int `$flags = 0` [, string `$class_name = 'HttpDeflateStream'`]])

public string **HttpDeflateStream::finish** ([string `$data`])

public string **HttpDeflateStream::flush** ([string `$data`])

public string **HttpDeflateStream::update** (string `$data`)
}

Class Members

Predefined Constants

Type	Name	Description
int	TYPE_GZIP	gzip encoding
int	TYPE_ZLIB	zlib AKA deflate encoding
int	TYPE_RAW	raw deflate encoding
int	LEVEL_DEF	default compression level
int	LEVEL_MIN	minimum compression level
int	LEVEL_MAX	maximum compression level
int	STRATEGY_DEF	default strategy
int	STRATEGY_FILT	filtered strategy

int	STRATEGY_HUFF	Huffman strategy
int	STRATEGY_RLE	RLE strategy
int	STRATEGY_FIXED	fixed strategy
int	FLUSH_NONE	no forced flush
int	FLUSH_SYNC	synching flush
int	FLUSH_FULL	full flush

Examples

Example #1 - A HttpDeflateStream example

```
<?php
$stream = new HttpDeflateStream(
   HttpDeflateStream::TYPE_GZIP |
   HttpDeflateStream::LEVEL_MAX |
   HttpDeflateStream::FLUSH_SYNC);

echo $stream->update($data);
echo $stream->finish();
?>
```

HttpDeflateStream::__construct

HttpDeflateStream::__construct -- HttpDeflateStream class constructor

Description

public void **HttpDeflateStream::__construct** ([int `$flags = 0`])

Creates a new HttpDeflateStream object instance.

See the deflate stream constants table for possible *`flags`*.

Parameters

`flags`
: initialization flags

See Also

- **HttpDeflateStream::factory**

HttpDeflateStream::factory

HttpDeflateStream::factory -- HttpDeflateStream class factory

Description

public HttpDeflateStream **HttpDeflateStream::factory** ([int `$flags = 0` [, string `$class_name = 'HttpDeflateStream'`]])

Creates a new HttpDeflateStream object instance.

See the deflate stream constants table for possible *`flags`*.

Parameters

`flags`
: initialization flags

`class_name`
: name of a subclass of HttpDeflateStream

See Also

- **HttpDeflateStream::__construct**

HttpDeflateStream::finish

HttpDeflateStream::finish -- Finalize deflate stream

Description

public string **HttpDeflateStream::finish** ([string `$data`])

Finalizes the deflate stream. The deflate stream can be reused after finalizing.

Parameters

`data`
: data to deflate

Return Values

Returns the final part of deflated data.

HttpDeflateStream::flush

HttpDeflateStream::flush -- Flush deflate stream

Description

public string **HttpDeflateStream::flush** ([string `$data`])

Flushes the deflate stream.

Parameters

`data`
more data to deflate

Return Values

Returns some deflated data as string on success or **FALSE** on failure.

HttpDeflateStream::update

HttpDeflateStream::update -- Update deflate stream

Description

public string **HttpDeflateStream::update** (string `$data`)

Passes more data through the deflate stream.

Parameters

`data`
: data to deflate

Return Values

Returns deflated data on success or **FALSE** on failure.

The HttpInflateStream class

Class synopsis

HttpInflateStream

HttpInflateStream {

public void **HttpInflateStream::__construct** ([int `$flags = 0`])

public HttpInflateStream **HttpInflateStream::factory** ([int `$flags = 0` [, string `$class_name = 'HttpInflateStream'`]])

public string **HttpInflateStream::finish** ([string `$data`])

public string **HttpInflateStream::flush** ([string `$data`])

public string **HttpInflateStream::update** (string `$data`)
}

Class Members

Constants

Type	Name	Description
int	FLUSH_NONE	no forced flush
int	FLUSH_SYNC	synching flush
int	FLUSH_FULL	full flush

Note

Flushing usually has no effect on inflate streams.

Examples

Example #2 - A HttpInflateStream example

```
<?php
$stream = new HttpInflateStream;
echo $stream->update($data);
echo $stream->finish();
?>
```

HttpInflateStream::__construct

HttpInflateStream::__construct -- HttpInflateStream class constructor

Description

public void **HttpInflateStream::__construct** ([int `$flags = 0`])

Creates a new HttpInflateStream object instance.

See the inflate constants table for possible *`flags`*.

Parameters

`flags`
: initialization flags

See Also

- **HttpInflateStream::factory**

HttpInflateStream::factory

HttpInflateStream::factory -- HttpInflateStream class factory

Description

public HttpInflateStream **HttpInflateStream::factory** ([int `$flags = 0` [, string `$class_name = 'HttpInflateStream'`]])

Creates a new HttpInflateStream object instance.

See the inflate constants table for possible *`flags`*.

Parameters

`flags`
: initialization flags

`class_name`
: name of a subclass of HttpInflateStream

See Also

- **HttpInflateStream::__construct**

HttpInflateStream::finish

HttpInflateStream::finish -- Finalize inflate stream

Description

public string **HttpInflateStream::finish** ([string `$data`])

Finalizes the inflate stream. The inflate stream can be reused after finalizing.

Parameters

`data`
: data to inflate

Return Values

Returns the final part of inflated data.

HttpInflateStream::flush

HttpInflateStream::flush -- Flush inflate stream

Description

public string **HttpInflateStream::flush** ([string `$data`])

Flushes the inflate stream.

Note
Flushing usually has no effect on inflate streams.

Parameters

`data`
: more data to inflate

Return Values

Returns some inflated data as string on success or **FALSE** on failure.

HttpInflateStream::update

HttpInflateStream::update -- Update inflate stream

Description

public string **HttpInflateStream::update** (string `$data`)

Passes more data through the inflate stream.

Parameters

`data`
: data to inflate

Return Values

Returns inflated data on success or **FALSE** on failure.

The HttpMessage class

Class synopsis

HttpMessage

HttpMessage implements Iterator, Countable, Serializable {

public void **HttpMessage::addHeaders** (array `$headers` [, bool `$append = FALSE`])

public void **HttpMessage::__construct** ([string `$message`])

public HttpMessage **HttpMessage::detach** (void)

static public HttpMessage **HttpMessage::factory** ([string `$raw_message` [, string `$class_name = 'HttpMessage'`]])

static public HttpMessage **HttpMessage::fromEnv** (int `$message_type` [, string `$class_name = 'HttpMessage'`])

static public HttpMessage **HttpMessage::fromString** ([string `$raw_message` [, string `$class_name = 'HttpMessage'`]])

public string **HttpMessage::getBody** (void)

public string **HttpMessage::getHeader** (string `$header`)

public array **HttpMessage::getHeaders** (void)

public string **HttpMessage::getHttpVersion** (void)

public HttpMessage **HttpMessage::getParentMessage** (void)

public string **HttpMessage::getRequestMethod** (void)

public string **HttpMessage::getRequestUrl** (void)

public int **HttpMessage::getResponseCode** (void)

public string **HttpMessage::getResponseStatus** (void)

public int **HttpMessage::getType** (void)

public string **HttpMessage::guessContentType** (string `$magic_file` [, int `$magic_mode = MAGIC_MIME`])

public void **HttpMessage::prepend** (HttpMessage `$message` [, bool `$top = TRUE`])

public HttpMessage **HttpMessage::reverse** (void)

public bool **HttpMessage::send** (void)

public void **HttpMessage::setBody** (string `$body`)

public void **HttpMessage::setHeaders** (array `$headers`)

public bool **HttpMessage::setHttpVersion** (string `$version`)

public bool **HttpMessage::setRequestMethod** (string `$method`)

public bool **HttpMessage::setRequestUrl** (string `$url`)

public bool **HttpMessage::setResponseCode** (int `$code`)

public bool **HttpMessage::setResponseStatus** (string `$status`)

public void **HttpMessage::setType** (int `$type`)

public HttpRequest|HttpResponse **HttpMessage::toMessageTypeObject** (void)

public string **HttpMessage::toString** ([bool `$include_parent = FALSE`])
}

Class Members

Properties

Instance Properties

Modifiers	Type	Name	Description
protected	int	type	message type
protected	string	body	message body
protected	float	httpVersion	HTTP protocol version
protected	array	headers	message headers
protected	string	requestMethod	request method name
protected	requestUrl	string	request URL

protected	int	responseCode	response code
protected	string	responseStatus	response status message
protected	HttpMessage	parentMessage	reference to parent message

Note
None of these default properties can be accessed by reference, array key/index notation nor be used in increment or decrement operations.

Predefined Constants

Type	**Name**	**Description**
int	TYPE_NONE	message has is of no specific type
int	TYPE_REQUEST	message is a request style HTTP message
int	TYPE_RESPONSE	message is a response style HTTP message

HttpMessage::addHeaders

HttpMessage::addHeaders -- Add headers

Description

public void **HttpMessage::addHeaders** (array `$headers` [, bool `$append = FALSE`])

Add headers. If append is true, headers with the same name will be separated, else overwritten.

Parameters

`headers`
: associative array containing the additional HTTP headers to add to the messages existing headers

`append`
: if true, and a header with the same name of one to add exists already, this respective header will be converted to an array containing both header values, otherwise it will be overwritten with the new header value

Return Values

Returns **TRUE** on success or **FALSE** on failure.

HttpMessage::__construct

HttpMessage::__construct -- HttpMessage constructor

Description

public void **HttpMessage::__construct** ([string `$message`])

Instantiate a new HttpMessage object.

The constructed object will actually represent the *last* message of the passed string. If there were prior messages, those can be accessed by HttpMessage:: **getParentMessage()**.

Parameters

`message`
: a single or several consecutive HTTP messages

Errors/Exceptions

Throws HttpMalformedHeaderException.

HttpMessage::detach

HttpMessage::detach -- Detach HttpMessage

Description

public HttpMessage **HttpMessage::detach** (void)

Returns a clone of an HttpMessage object detached from any parent messages.

Parameters

Return Values

Returns detached HttpMessage object copy.

HttpMessage::factory

HttpMessage::factory -- Create HttpMessage from string

Description

static public HttpMessage **HttpMessage::factory** ([string `$raw_message` [, string `$class_name = 'HttpMessage'`]])

Create an HttpMessage object from a string.

Parameters

`raw_message`
: a single or several consecutive HTTP messages

`class_name`
: a class extending HttpMessage

Return Values

Returns an HttpMessage object on success or NULL on failure.

Errors/Exceptions

Throws HttpMalformedHeadersException.

See Also

- HttpMessage::fromEnv().

HttpMessage::fromEnv

HttpMessage::fromEnv -- Create HttpMessage from environment

Description

static public HttpMessage **HttpMessage::fromEnv** (int `$message_type` [, string `$class_name = 'HttpMessage'`])

Create an HttpMessage object from script environment.

Parameters

`message_type`
: The message type. See HttpMessage type constants.

`class_name`
: a class extending HttpMessage

Return Values

Returns an HttpMessage object on success or NULL on failure.

See Also

- HttpMessage::factory().

HttpMessage::fromString

HttpMessage::fromString -- Create HttpMessage from string

Description

static public HttpMessage **HttpMessage::fromString** ([string `$raw_message` [, string `$class_name = 'HttpMessage'`]])

Create an HttpMessage object from a string.

This function alias is deprecated and only exists for backwards compatibility reasons. The use of this function is not recommended, as it may be removed from PHP in the future.

Parameters

`raw_message`
: a single or several consecutive HTTP messages

`class_name`
: a class extending HttpMessage

Return Values

Returns an HttpMessage object on success or NULL on failure.

Errors/Exceptions

Throws HttpMalformedHeadersException.

See Also

- HttpMessage::factory().

HttpMessage::getBody

HttpMessage::getBody -- Get message body

Description

public string **HttpMessage::getBody** (void)

Get the body of the parsed HttpMessage.

Return Values

Returns the message body as string.

HttpMessage::getHeader

HttpMessage::getHeader -- Get header

Description

public string **HttpMessage::getHeader** (string `$header`)

Get message header.

Parameters

`header`
: header name

Return Values

Returns the header value on success or NULL if the header does not exist.

HttpMessage::getHeaders

HttpMessage::getHeaders -- Get message headers

Description

public array **HttpMessage::getHeaders** (void)

Get message headers.

Return Values

Returns an associative array containing the messages HTTP headers.

HttpMessage::getHttpVersion

HttpMessage::getHttpVersion -- Get HTTP version

Description

public string **HttpMessage::getHttpVersion** (void)

Get the HTTP Protocol Version of the Message.

Return Values

Returns the HTTP protocol version as string.

HttpMessage::getParentMessage

HttpMessage::getParentMessage -- Get parent message

Description

public HttpMessage **HttpMessage::getParentMessage** (void)

Get parent Message.

Return Values

Returns the parent HttpMessage object.

Errors/Exceptions

Throws HttpRuntimeException.

HttpMessage::getRequestMethod

HttpMessage::getRequestMethod -- Get request method

Description

public string **HttpMessage::getRequestMethod** (void)

Get the Request Method of the Message.

Return Values

Returns the request method name on success, or FALSE if the message is not of type HttpMessage:: **TYPE_REQUEST**.

HttpMessage::getRequestUrl

HttpMessage::getRequestUrl -- Get request URL

Description

public string **HttpMessage::getRequestUrl** (void)

Get the Request URL of the Message.

Parameters

Return Values

Returns the request URL as string on success, or FALSE if the message is not of type HttpMessage:: **TYPE_REQUEST**.

HttpMessage::getResponseCode

HttpMessage::getResponseCode -- Get response code

Description

public int **HttpMessage::getResponseCode** (void)

Get the Response Code of the Message.

Return Values

Returns the HTTP response code if the message is of type HttpMessage::
TYPE_RESPONSE, else FALSE.

HttpMessage::getResponseStatus

HttpMessage::getResponseStatus -- Get response status

Description

public string **HttpMessage::getResponseStatus** (void)

Get the Response Status of the message (i.e. the string following the response code).

Return Values

Returns the HTTP response status string if the message is of type HttpMessage:: **TYPE_RESPONSE**, else FALSE.

HttpMessage::getType

HttpMessage::getType -- Get message type

Description

public int **HttpMessage::getType** (void)

Get Message Type. Either **HTTP_MSG_NONE**, **HTTP_MSG_REQUEST** or **HTTP_MSG_RESPONSE**.

Return Values

Returns the HttpMessage:: **TYPE**.

HttpMessage::guessContentType

HttpMessage::guessContentType -- Guess content type

Description

public string **HttpMessage::guessContentType** (string `$magic_file` [, int `$magic_mode = MAGIC_MIME`])

Attempts to guess the content type of the message body through libmagic.

Parameters

`magic_file`
: the magic.mime database to use

`magic_mode`
: flags for libmagic

Return Values

Returns the guessed content type on success, or FALSE on failure.

Errors/Exceptions

Throws HttpRuntimeException, HttpInvalidParamException.

HttpMessage::prepend

HttpMessage::prepend -- Prepend message(s)

Description

public void **HttpMessage::prepend** (HttpMessage `$message` [, bool `$top = TRUE`])

Prepends message(s) to the HTTP message.

Parameters

`message`
: HttpMessage object to prepend

`top`
: whether to prepend to the top most or right this message

Errors/Exceptions

Throws HttpInvalidParamException if the message is located within the same message chain.

HttpMessage::reverse

HttpMessage::reverse -- Reverse message chain

Description

public HttpMessage **HttpMessage::reverse** (void)

Reorders the message chain in reverse order.

Return Values

Returns the most parent HttpMessage object.

HttpMessage::send

HttpMessage::send -- Send message

Description

public bool **HttpMessage::send** (void)

Send the Message according to its type as Response or Request.

This provides limited functionality compared to HttpRequest and HttpResponse.

Return Values

Returns **TRUE** on success or **FALSE** on failure.

HttpMessage::setBody

HttpMessage::setBody -- Set message body

Description

public void **HttpMessage::setBody** (string `$body`)

Set the body of the HttpMessage.

Note
Don't forget to update any headers accordingly.

Parameters

body
: the new body of the message

HttpMessage::setHeaders

HttpMessage::setHeaders -- Set headers

Description

public void **HttpMessage::setHeaders** (array `$headers`)

Sets new headers.

Parameters

`headers`
: associative array containing the new HTTP headers, which will replace *all* previous HTTP headers of the message

HttpMessage::setHttpVersion

HttpMessage::setHttpVersion -- Set HTTP version

Description

public bool **HttpMessage::setHttpVersion** (string `$version`)

Set the HTTP Protocol version of the Message.

Parameters

`version`
: the HTTP protocol version

Return Values

Returns TRUE on success, or FALSE if supplied version is out of range (1.0/1.1).

HttpMessage::setRequestMethod

HttpMessage::setRequestMethod -- Set request method

Description

public bool **HttpMessage::setRequestMethod** (string `$method`)

Set the Request Method of the HTTP Message.

Parameters

`method`
: the request method name

Return Values

Returns TRUE on success, or FALSE if the message is not of type HttpMessage::**TYPE_REQUEST** or an invalid request method was supplied.

HttpMessage::setRequestUrl

HttpMessage::setRequestUrl -- Set request URL

Description

public bool **HttpMessage::setRequestUrl** (string `$url`)

Set the Request URL of the HTTP Message.

Parameters

`url`
: the request URL

Return Values

Returns TRUE on success, or FALSE if the message is not of type HttpMessage::**TYPE_REQUEST** or supplied URL was empty.

HttpMessage::setResponseCode

HttpMessage::setResponseCode -- Set response code

Description

public bool **HttpMessage::setResponseCode** (int `$code`)

Set the response code of an HTTP Response Message.

Parameters

code
HTTP response code

Return Values

Returns TRUE on success, or FALSE if the message is not of type HttpMessage::**TYPE_RESPONSE** or the response code is out of range (100-510).

HttpMessage::setResponseStatus

HttpMessage::setResponseStatus -- Set response status

Description

public bool **HttpMessage::setResponseStatus** (string `$status`)

Set the Response Status of the HTTP message (i.e. the string following the response code).

Parameters

`status`
: the response status text

Return Values

Returns TRUE on success or FALSE if the message is not of type HttpMessage::**TYPE_RESPONSE**.

HttpMessage::setType

HttpMessage::setType -- Set message type

Description

public void **HttpMessage::setType** (int `$type`)

Set Message Type. Either **HTTP_MSG_NONE**, **HTTP_MSG_REQUEST** or **HTTP_MSG_RESPONSE**.

Parameters

`type`
: the HttpMessage::TYPE

HttpMessage::toMessageTypeObject

HttpMessage::toMessageTypeObject -- Create HTTP object regarding message type

Description

public HttpRequest|HttpResponse **HttpMessage::toMessageTypeObject** (void)

Creates an object regarding to the type of the message.

Parameters

Return Values

Returns either an HttpRequest or HttpResponse object on success, or NULL on failure.

Errors/Exceptions

Throws HttpRuntimeException, HttpMessageTypeException, HttpHeaderException.

HttpMessage::toString

HttpMessage::toString -- Get string representation

Description

public string **HttpMessage::toString** ([bool `$include_parent = FALSE`])

Get the string representation of the Message.

Parameters

`include_parent`
: specifies whether the returned string should also contain any parent messages

Return Values

Returns the message as string.

The HttpQueryString class

Class synopsis

HttpQueryString

HttpQueryString implements ArrayAccess, Serializable {

final public void **HttpQueryString::__construct** ([bool `$global = TRUE` [, mixed `$add`]])

public mixed **HttpQueryString::get** ([string `$key` [, mixed `$type = 0` [, mixed `$defval = NULL` [, bool `$delete = FALSE`]]]])

public HttpQueryString **HttpQueryString::mod** (mixed `$params`)

public string **HttpQueryString::set** (mixed `$params`)

static public HttpQueryString **HttpQueryString::singleton** ([bool `$global = TRUE`])

public array **HttpQueryString::toArray** (void)

public string **HttpQueryString::toString** (void)

public bool **HttpQueryString::xlate** (string `$ie`, string `$oe`)
}

Class Members

Properties

Instance Properties

Modifiers	Type	Name	Description
private	array	queryArray	query parameters
private	string	queryString	serialized query parameters

Static Properties

Modifiers	Type	Name	Descriptio
private	array	instance	holds sing

Predefined Constants

Type	Name	Description
int	TYPE_BOOL	retrieve query param as bool
int	TYPE_INT	retrieve query param as int
int	TYPE_FLOAT	retrieve query param as float
int	TYPE_STRING	retrieve query param as string
int	TYPE_ARRAY	retrieve query param as array
int	TYPE_OBJECT	retrieve query param as object

HttpQueryString::__construct

HttpQueryString::__construct -- HttpQueryString constructor

Description

final public void **HttpQueryString::__construct** ([bool `$global = TRUE` [, mixed `$add`]])

Creates a new HttpQueryString object instance.

Operates on and modifies $_GET and $_SERVER['QUERY_STRING'] if global is TRUE.

Parameters

`global`
: whether to operate on $_GET and $_SERVER['QUERY_STRING']

`add`
: additional/initial query string parameters

Errors/Exceptions

Throws HttpRuntimeException.

HttpQueryString::get

HttpQueryString::get -- Get (part of) query string

Description

public mixed **HttpQueryString::get** ([string `$key` [, mixed `$type = 0` [, mixed `$defval = NULL` [, bool `$delete = FALSE`]]]])

Get (part of) the query string.

The type parameter is either one of the HttpQueryString::TYPE_* constants or a type abbreviation like "b" for bool, "i" for int, "f" for float, "s" for string, "a" for array and "o" for a stdClass object.

Parameters

`key`
: key of the query string param to retrieve

`type`
: which variable type to enforce

`defval`
: default value if key does not exist

`delete`
: whether to remove the key/value pair from the query string

Return Values

Returns the value of the query string param or the whole query string if no key was specified on success or defval if key does not exist.

HttpQueryString::mod

HttpQueryString::mod -- Modifiy query string copy

Description

public HttpQueryString **HttpQueryString::mod** (mixed `$params`)

Copies the query string object and sets provided params at the clone.

Parameters

`params`
: query string params to add

Return Values

Returns a new HttpQueryString object

HttpQueryString::set

HttpQueryString::set -- Set query string params

Description

public string **HttpQueryString::set** (mixed `$params`)

Set query string entry/entries. **NULL** values will unset the variable.

Parameters

`params`
query string params to add

Return Values

Returns the current query string.

HttpQueryString::singleton

HttpQueryString::singleton -- HttpQueryString singleton

Description

static public HttpQueryString **HttpQueryString::singleton** ([bool `$global = TRUE`])

Get a single instance (differentiates between the global setting).

Parameters

`global`
: whether to operate on $_GET and $_SERVER['QUERY_STRING']

Return Values

Returns always the same HttpQueryString instance regarding the global setting.

Errors/Exceptions

Throws HttpRuntimeException.

HttpQueryString::toArray

HttpQueryString::toArray -- Get query string as array

Description

public array **HttpQueryString::toArray** (void)

Get the query string represented as associative array.

Return Values

Returns the array representation of the query string.

HttpQueryString::toString

HttpQueryString::toString -- Get query string

Description

public string **HttpQueryString::toString** (void)

Get the query string.

Parameters

Return Values

Returns the string representation of the query string.

HttpQueryString::xlate

HttpQueryString::xlate -- Change query strings charset

Description

public bool **HttpQueryString::xlate** (string `$ie`, string `$oe`)

Converts the query string from the source encoding ie to the target encoding oe.

Warning
Don't use any character set that can contain NUL bytes like UTF-16.

Note
This method requires ext/iconv to be enabled and loaded.

Parameters

`ie`
: input encoding

`oe`
: output encoding

Return Values

Returns **TRUE** on success or **FALSE** on failure.

The HttpRequest

Class synopsis

HttpRequest

HttpRequest {

public bool **HttpRequest::addCookies** (array `$cookies`)

public bool **HttpRequest::addHeaders** (array `$headers`)

public bool **HttpRequest::addPostFields** (array `$post_data`)

public bool **HttpRequest::addPostFile** (string `$name`, string `$file` [, string `$content_type = 'application/x-octetstream'`])

public bool **HttpRequest::addPutData** (string `$put_data`)

public bool **HttpRequest::addQueryData** (array `$query_params`)

public bool **HttpRequest::addRawPostData** (string `$raw_post_data`)

public bool **HttpRequest::addSslOptions** (array `$options`)

public void **HttpRequest::clearHistory** (void)

public void **HttpRequest::__construct** ([string `$url` [, int `$request_method = HTTP_METH_GET` [, array `$options`]]])

public bool **HttpRequest::enableCookies** (void)

public string **HttpRequest::getContentType** (void)

public array **HttpRequest::getCookies** (void)

public array **HttpRequest::getHeaders** (void)

public HttpMessage **HttpRequest::getHistory** (void)

public int **HttpRequest::getMethod** (void)

public array **HttpRequest::getOptions** (void)

public array **HttpRequest::getPostFields** (void)

public array **HttpRequest::getPostFiles** (void)

public string **HttpRequest::getPutData** (void)

public string **HttpRequest::getPutFile** (void)

public string **HttpRequest::getQueryData** (void)

public string **HttpRequest::getRawPostData** (void)

public string **HttpRequest::getRawRequestMessage** (void)

public string **HttpRequest::getRawResponseMessage** (void)

public HttpMessage **HttpRequest::getRequestMessage** (void)

public string **HttpRequest::getResponseBody** (void)

public int **HttpRequest::getResponseCode** (void)

public array **HttpRequest::getResponseCookies** ([int `$flags` [, array `$allowed_extras`]])

public array **HttpRequest::getResponseData** (void)

public mixed **HttpRequest::getResponseHeader** ([string `$name`])

public mixed **HttpRequest::getResponseInfo** ([string `$name`])

public HttpMessage **HttpRequest::getResponseMessage** (void)

public string **HttpRequest::getResponseStatus** (void)

public array **HttpRequest::getSslOptions** (void)

public string **HttpRequest::getUrl** (void)

public bool **HttpRequest::resetCookies** ([bool `$session_only = FALSE`])

public HttpMessage **HttpRequest::send** (void)

public bool **HttpRequest::setContentType** (string `$content_type`)

public bool **HttpRequest::setCookies** ([array `$cookies`])

public bool **HttpRequest::setHeaders** ([array `$headers`])

public bool **HttpRequest::setMethod** (int `$request_method`)

public bool **HttpRequest::setOptions** ([array `$options`])

public bool **HttpRequest::setPostFields** (array `$post_data`)

public bool **HttpRequest::setPostFiles** (array `$post_files`)

public bool **HttpRequest::setPutData** ([string `$put_data`])

public bool **HttpRequest::setPutFile** ([string `$file`])

public bool **HttpRequest::setQueryData** (mixed `$query_data`)

public bool **HttpRequest::setRawPostData** ([string `$raw_post_data`])

public bool **HttpRequest::setSslOptions** ([array `$options`])

public bool **HttpRequest::setUrl** (string `$url`)
}

Class Members

Properties

Instance Properties

Modifiers	Type	Name	Description
protected	array	options	request options to configure the request; see request options
protected	array	postFields	form data: *array("fieldname" => "fieldvalue")*
protected	array	postFiles	files to upload: *array(array("name" => "image", "file" => "/home/u/images/u.png", "type" => "image/png"))*
protected	array	responseInfo	information (statistical) about the request/response; see Request/response information
protected	HttpMessage	responseMessage	the response message

protected	integer	responseCode	the numerical response code
protected	string	responseStatus	the literal response status text
protected	integer	method	the request method to use
protected	string	url	the request url
protected	string	contentType	the content type to use for raw post requests
protected	string	rawPostData	raw post data
protected	string	queryData	query parameters
protected	string	putFile	the file to upload with a PUT request
protected	string	putData	raw data to upload with a PUT request
protected	HttpMessage	history	the whole request/response history if history logging is enabled
public	boolean	recordHistory	whether to enable history logging

Predefined Constants

Type	Name	Description
integer	METH_GET	GET request method
integer	METH_HEAD	HEAD request method
integer	METH_POST	POST request method
integer	METH_PUT	PUT request method
integer	METH_DELETE	DELETE request method
integer	METH_OPTIONS	OPTIONS request method

integer	METH_TRACE	TRACE request method
integer	METH_CONNECT	CONNECT request method
integer	METH_PROPFIND	PROPFIND request method
integer	METH_PROPPATCH	PROPPATCH request method
integer	METH_MKCOL	MKCOL request method
integer	METH_COPY	COPY request method
integer	METH_MOVE	MOVE request method
integer	METH_LOCK	LOCK request method
integer	METH_UNLOCK	UNLOCK request method
integer	METH_VERSION_CONTROL	VERSION-CONTROL request method
integer	METH_REPORT	REPORT request method
integer	METH_CHECKOUT	CHECKOUT request method
integer	METH_CHECKIN	CHECKIN request method
integer	METH_UNCHECKOUT	UNCHECKOUT request method
integer	METH_MKWORKSPACE	MKWORKSPACE request method
integer	METH_UPDATE	UPDATE request method
integer	METH_LABEL	LABEL request method
integer	METH_MERGE	MERGE request method
integer	METH_BASELINE_CONTROL	BASELINE-CONTROL request method
integer	METH_MKACTIVITY	MKACTIVITY request method
integer	METH_ACL	ACL request method
integer	VERSION_1_0	HTTP protocol version 1.0
integer	VERSION_1_1	HTTP protocol version 1.1

integer	VERSION_ANY	any HTTP protocol version
integer	AUTH_BASIC	basic authentication
integer	AUTH_DIGEST	digest authentication
integer	AUTH_NTLM	NTLM authentication
integer	AUTH_GSSNEG	GSS negotiate authentication
integer	AUTH_ANY	any authentication
integer	PROXY_SOCKS4	SOCKS v4 proxy
integer	PROXY_SOCKS5	SOCKS v5 proxy
integer	PROXY_HTTP	HTTP proxy
integer	SSL_VERSION_TLSv1	use TLS v1
integer	SSL_VERSION_SSLv2	use SSL v2
integer	SSL_VERSION_SSLv3	use SSL v3
integer	SSL_VERSION_ANY	use any SSL/TLS method
integer	IPRESOLVE_V4	resolve via IPv4 only
integer	IPRESOLVE_V6	resolve via IPv6 only
integer	IPRESOLVE_ANY	use any resolving methods

HttpRequest::addCookies

HttpRequest::addCookies -- Add cookies

Description

public bool **HttpRequest::addCookies** (array `$cookies`)

Add custom cookies.

Note
The request option *encodecookies* controls whether the cookie values should be urlencode() d.

Note
Affects any request method.

Parameters

`cookies`
: an associative array containing any cookie name/value pairs to add

Return Values

Returns **TRUE** on success or **FALSE** on failure.

Examples

Example #3 - A HttpRequest::addCookies() example

```
<?php
$r = new HttpRequest;
$r->addCookies(
   array(
       "cookie_name" => "cookie value",
   )
);
?>
```

See Also

- HttpRequest::setCookies()

HttpRequest::addHeaders

HttpRequest::addHeaders -- Add headers

Description

public bool **HttpRequest::addHeaders** (array `$headers`)

Add request header name/value pairs.

Parameters

`headers`
: an associative array as parameter containing additional header name/value pairs

Return Values

Returns **TRUE** on success or **FALSE** on failure.

HttpRequest::addPostFields

HttpRequest::addPostFields -- Add post fields

Description

public bool **HttpRequest::addPostFields** (array `$post_data`)

Adds POST data entries, leaving previously set unchanged, unless a post entry with the same name already exists.

Affects only POST and custom requests.

Parameters

`post_data`
: an associative array as parameter containing the post fields

Return Values

Returns **TRUE** on success or **FALSE** on failure.

HttpRequest::addPostFile

HttpRequest::addPostFile -- Add post file

Description

public bool **HttpRequest::addPostFile** (string `$name`, string `$file` [, string `$content_type = 'application/x-octetstream'`])

Add a file to the POST request, leaving previously set files unchanged.

Affects only POST and custom requests. Cannot be used with raw post data.

Parameters

`name`
: the form element name

`file`
: the path to the file

`content_type`
: the content type of the file

Return Values

Returns TRUE on success, or FALSE if the content type seems not to contain a primary and a secondary content type part.

HttpRequest::addPutData

HttpRequest::addPutData -- Add put data

Description

public bool **HttpRequest::addPutData** (string `$put_data`)

Add PUT data, leaving previously set PUT data unchanged.

Affects only PUT requests.

Parameters

`put_data`
: the data to concatenate

Return Values

Returns **TRUE** on success or **FALSE** on failure.

HttpRequest::addQueryData

HttpRequest::addQueryData -- Add query data

Description

public bool **HttpRequest::addQueryData** (array `$query_params`)

Add parameters to the query parameter list, leaving previously set unchanged.

Affects any request type.

Parameters

`query_params`
: an associative array as parameter containing the query fields to add

Return Values

Returns **TRUE** on success or **FALSE** on failure.

HttpRequest::addRawPostData

HttpRequest::addRawPostData -- Add raw post data

Description

public bool **HttpRequest::addRawPostData** (string `$raw_post_data`)

Add raw post data, leaving previously set raw post data unchanged.

Affects only POST and custom requests.

Parameters

`raw_post_data`
: the raw post data to concatenate

Return Values

Returns **TRUE** on success or **FALSE** on failure.

HttpRequest::addSslOptions

HttpRequest::addSslOptions -- Add ssl options

Description

public bool **HttpRequest::addSslOptions** (array `$options`)

Set additional SSL options.

Parameters

`options`
: an associative array as parameter containing additional SSL specific options

Return Values

Returns **TRUE** on success or **FALSE** on failure.

HttpRequest::clearHistory

HttpRequest::clearHistory -- Clear history

Description

public void **HttpRequest::clearHistory** (void)

Clears all history messages.

Note
History is only logged if *recordHistory* was enabled.

HttpRequest::__construct

HttpRequest::__construct -- HttpRequest constructor

Description

public void **HttpRequest::__construct** ([string `$url` [, int `$request_method = HTTP_METH_GET` [, array `$options`]]])

Instantiate a new HttpRequest object.

Parameters

`url`
: the target request url

`request_method`
: the request method to use

`options`
: an associative array with request options

Errors/Exceptions

Throws HttpException.

HttpRequest::enableCookies

HttpRequest::enableCookies -- Enable cookies

Description

public bool **HttpRequest::enableCookies** (void)

Enable automatic sending of received cookies.

Note
Note that cuutomly set cookies will be sent anyway.

Return Values

Returns **TRUE** on success or **FALSE** on failure.

HttpRequest::getContentType

HttpRequest::getContentType -- Get content type

Description

public string **HttpRequest::getContentType** (void)

Get the previously set content type.

Return Values

Returns the previously set content type as string.

HttpRequest::getCookies

HttpRequest::getCookies -- Get cookies

Description

public array **HttpRequest::getCookies** (void)

Get previously set cookies.

Return Values

Returns an associative array containing any previously set cookies.

HttpRequest::getHeaders

HttpRequest::getHeaders -- Get headers

Description

public array **HttpRequest::getHeaders** (void)

Get previously set request headers.

Return Values

Returns an associative array containing all currently set headers.

HttpRequest::getHistory

HttpRequest::getHistory -- Get history

Description

public HttpMessage **HttpRequest::getHistory** (void)

Get all sent requests and received responses as an HttpMessage object.

If you want to record history, set the instance variable HttpRequest:: *recordHistory* to TRUE.

The returned object references the last received response, use **HttpMessage::getParentMessage()()** to access the data of previously sent requests and received responses.

Return Values

Returns an HttpMessage object representing the complete request/response history.

Errors/Exceptions

Throws HttpRuntimeException.

HttpRequest::getMethod

HttpRequest::getMethod -- Get method

Description

public int **HttpRequest::getMethod** (void)

Get the previously set request method.

Return Values

Returns the currently set request method.

HttpRequest::getOptions

HttpRequest::getOptions -- Get options

Description

public array **HttpRequest::getOptions** (void)

Get currently set options.

Return Values

Returns an associative array containing currently set options.

HttpRequest::getPostFields

HttpRequest::getPostFields -- Get post fields

Description

public array **HttpRequest::getPostFields** (void)

Get previously set POST data.

Parameters

Return Values

Returns the currently set post fields as associative array.

HttpRequest::getPostFiles

HttpRequest::getPostFiles -- Get post files

Description

public array **HttpRequest::getPostFiles** (void)

Get all previously added POST files.

Return Values

Returns an array containing currently set post files.

HttpRequest::getPutData

HttpRequest::getPutData -- Get put data

Description

public string **HttpRequest::getPutData** (void)

Get previously set PUT data.

Return Values

Returns a string containing the currently set PUT data.

HttpRequest::getPutFile

HttpRequest::getPutFile -- Get put file

Description

public string **HttpRequest::getPutFile** (void)

Get previously set put file.

Return Values

Returns a string containing the path to the currently set put file.

HttpRequest::getQueryData

HttpRequest::getQueryData -- Get query data

Description

public string **HttpRequest::getQueryData** (void)

Get the current query data in form of an urlencoded query string.

Return Values

Returns a string containing the urlencoded query.

HttpRequest::getRawPostData

HttpRequest::getRawPostData -- Get raw post data

Description

public string **HttpRequest::getRawPostData** (void)

Get previously set raw post data.

Parameters

Return Values

Returns a string containing the currently set raw post data.

HttpRequest::getRawRequestMessage

HttpRequest::getRawRequestMessage -- Get raw request message

Description

public string **HttpRequest::getRawRequestMessage** (void)

Get sent HTTP message.

Parameters

Return Values

Returns an HttpMessage in a form of a string.

HttpRequest::getRawResponseMessage

HttpRequest::getRawResponseMessage -- Get raw response message

Description

public string **HttpRequest::getRawResponseMessage** (void)

Get the entire HTTP response.

Parameters

Return Values

Returns the complete web server response, including the headers in a form of a string.

HttpRequest::getRequestMessage

HttpRequest::getRequestMessage -- Get request message

Description

public HttpMessage **HttpRequest::getRequestMessage** (void)

Get sent HTTP message.

If redirects were allowed and several responses were received, the data references the last received response. Use HttpMessage::getParentMessage() to access the data of previously sent requests within this request cycle.

Note
Note that the internal request message is immutable, that means that the request message received through HttpRequest::getRequestMessage() will always look the same for the same request, regardless of any changes you may have made to the returned object.

Return Values

Returns an HttpMessage object representing the sent request.

Errors/Exceptions

Throws HttpMalformedHeadersException, HttpEncodingException.

HttpRequest::getResponseBody

HttpRequest::getResponseBody -- Get response body

Description

public string **HttpRequest::getResponseBody** (void)

Get the response body after the request has been sent.

If redirects were allowed and several responses were received, the data references the last received response.

Parameters

Return Values

Returns a string containing the response body.

HttpRequest::getResponseCode

HttpRequest::getResponseCode -- Get response code

Description

public int **HttpRequest::getResponseCode** (void)

Get the response code after the request has been sent.

If redirects were allowed and several responses were received, the data references the last received response.

Return Values

Returns an int representing the response code.

HttpRequest::getResponseCookies

HttpRequest::getResponseCookies -- Get response cookie(s)

Description

public array **HttpRequest::getResponseCookies** ([int `$flags` [, array `$allowed_extras`]])

Get response cookie(s) after the request has been sent.

If redirects were allowed and several responses were received, the data references the last received response.

Parameters

`flags`
: http_parse_cookie() flags

`allowed_extras`
: allowed keys treated as extra information instead of cookie names

Return Values

Returns an array of stdClass objects like http_parse_cookie() would return.

HttpRequest::getResponseData

HttpRequest::getResponseData -- Get response data

Description

public array **HttpRequest::getResponseData** (void)

* Get all response data after the request has been sent.

If redirects were allowed and several responses were received, the data references the last received response.

Return Values

Returns an associative array with the key "headers" containing an associative array holding all response headers, as well as the key "body" containing a string with the response body.

HttpRequest::getResponseHeader

HttpRequest::getResponseHeader -- Get response header(s)

Description

public mixed **HttpRequest::getResponseHeader** ([string `$name`])

Get response header(s) after the request has been sent.

If redirects were allowed and several responses were received, the data references the last received response.

Parameters

`name`
: header to read; if empty, all response headers will be returned

Return Values

Returns either a string with the value of the header matching name if requested, FALSE on failure, or an associative array containing all response headers.

HttpRequest::getResponseInfo

HttpRequest::getResponseInfo -- Get response info

Description

public mixed **HttpRequest::getResponseInfo** ([string `$name`])

Get response info after the request has been sent.

See http_get() for a full list of returned info.

If redirects were allowed and several responses were received, the data references the last received response.

Parameters

`name`
: the info to read; if empty or omitted, an associative array containing all available info will be returned

Return Values

Returns either a scalar containing the value of the info matching name if requested, FALSE on failure, or an associative array containing all available info.

HttpRequest::getResponseMessage

HttpRequest::getResponseMessage -- Get response message

Description

public HttpMessage **HttpRequest::getResponseMessage** (void)

Get the full response as HttpMessage object after the request has been sent.

If redirects were allowed and several responses were received, the data references the last received response. Use HttpMessage::getParentMessage() to access the data of previously received responses within this request cycle.

Return Values

Returns an HttpMessage object of the response.

Errors/Exceptions

Throws HttpException, HttpRuntimeException.

HttpRequest::getResponseStatus

HttpRequest::getResponseStatus -- Get response status

Description

public string **HttpRequest::getResponseStatus** (void)

Get the response status (i.e. the string after the response code) after the message has been sent.

Return Values

Returns a string containing the response status text.

HttpRequest::getSslOptions

HttpRequest::getSslOptions -- Get ssl options

Description

public array **HttpRequest::getSslOptions** (void)

Get previously set SSL options.

Return Values

Returns an associative array containing any previously set SSL options.

HttpRequest::getUrl

HttpRequest::getUrl -- Get url

Description

public string **HttpRequest::getUrl** (void)

Get the previously set request URL.

Return Values

Returns the currently set request url as string.

HttpRequest::resetCookies

HttpRequest::resetCookies -- Reset cookies

Description

public bool **HttpRequest::resetCookies** ([bool `$session_only = FALSE`])

Reset all automatically received/sent cookies.

Note
Note that customly set cookies are not affected.

Parameters

`session_only`
: whether only session cookies should be reset (needs libcurl >= v7.15.4, else libcurl >= v7.14.1)

Return Values

Returns **TRUE** on success or **FALSE** on failure.

HttpRequest::send

HttpRequest::send -- Send request

Description

public HttpMessage **HttpRequest::send** (void)

Send the HTTP request.

Note
While an exception may be thrown, the transfer could have succeeded at least partially, so you might want to check the return values of various HttpRequest::getResponse*() methods.

Return Values

Returns the received response as HttpMessage object.

Errors/Exceptions

Throws HttpRuntimeException, HttpRequestException, HttpMalformedHeaderException, HttpEncodingException.

Examples

Example #4 - GET example

```
<?php
$r = new HttpRequest('http://example.com/feed.rss', HttpRequest::METH_GET);
$r->setOptions(array('lastmodified' => filemtime('local.rss')));
$r->addQueryData(array('category' => 3));
try {
   $r->send();
   if ($r->getResponseCode() == 200) {
       file_put_contents('local.rss', $r->getResponseBody());
   }
} catch (HttpException $ex) {
   echo $ex;
}
?>
```

Example #5 - POST example

```
<?php
$r = new HttpRequest('http://example.com/form.php', HttpRequest::METH_POST);
$r->setOptions(array('cookies' => array('lang' => 'de')));
$r->addPostFields(array('user' => 'mike', 'pass' => 's3c|r3t'));
$r->addPostFile('image', 'profile.jpg', 'image/jpeg');
try {
   echo $r->send()->getBody();
} catch (HttpException $ex) {
   echo $ex;
}
?>
```

HttpRequest::setContentType

HttpRequest::setContentType -- Set content type

Description

public bool **HttpRequest::setContentType** (string `$content_type`)

Set the content type the post request should have.

Parameters

`content_type`
: the content type of the request (primary/secondary)

Return Values

Returns TRUE on success, or FALSE if the content type does not seem to contain a primary and a secondary part.

HttpRequest::setCookies

HttpRequest::setCookies -- Set cookies

Description

public bool **HttpRequest::setCookies** ([array `$cookies`])

Set custom cookies.

Parameters

`cookies`
: an associative array as parameter containing cookie name/value pairs; if empty or omitted, all previously set cookies will be unset

Return Values

Returns **TRUE** on success or **FALSE** on failure.

HttpRequest::setHeaders

HttpRequest::setHeaders -- Set headers

Description

public bool **HttpRequest::setHeaders** ([array `$headers`])

Set request header name/value pairs.

Parameters

`headers`
: an associative array as parameter containing header name/value pairs; if empty or omitted, all previously set headers will be unset

Return Values

Returns **TRUE** on success or **FALSE** on failure.

HttpRequest::setMethod

HttpRequest::setMethod -- Set method

Description

public bool **HttpRequest::setMethod** (int `$request_method`)

Set the request method.

Parameters

`request_method`
: the request method to use

Return Values

Returns **TRUE** on success or **FALSE** on failure.

HttpRequest::setOptions

HttpRequest::setOptions -- Set options

Description

public bool **HttpRequest::setOptions** ([array `$options`])

Set the request options to use.

See the full list of request options.

Parameters

`options`
: an associative array, which values will overwrite the currently set request options; if empty or omitted, the options of the HttpRequest object will be reset

Return Values

Returns **TRUE** on success or **FALSE** on failure.

HttpRequest::setPostFields

HttpRequest::setPostFields -- Set post fields

Description

public bool **HttpRequest::setPostFields** (array `$post_data`)

Set the POST data entries, overwriting previously set POST data.

Affects only POST and custom requests.

Parameters

`post_data`
: an associative array containing the post fields; if empty, the post data will be unset

Return Values

Returns **TRUE** on success or **FALSE** on failure.

HttpRequest::setPostFiles

HttpRequest::setPostFiles -- Set post files

Description

public bool **HttpRequest::setPostFiles** (array `$post_files`)

Set files to post, overwriting previously set post files.

Affects only POST and requests. Cannot be used with raw post data.

Parameters

`post_files`
: an array containing the files to post; if empty, the post files will be unset

Return Values

Returns **TRUE** on success or **FALSE** on failure.

HttpRequest::setPutData

HttpRequest::setPutData -- Set put data

Description

public bool **HttpRequest::setPutData** ([string `$put_data`])

Set PUT data to send, overwriting previously set PUT data.

Affects only PUT requests.

Only either PUT data or PUT file can be used for each request. PUT data has higher precedence and will be used even if a PUT file is set.

Parameters

`put_data`
: the data to upload

Return Values

Returns **TRUE** on success or **FALSE** on failure.

HttpRequest::setPutFile

HttpRequest::setPutFile -- Set put file

Description

public bool **HttpRequest::setPutFile** ([string `$file`])

Set file to put. Affects only PUT requests.

Parameters

`file`
: the path to the file to send; if empty or omitted the put file will be unset

Return Values

Returns **TRUE** on success or **FALSE** on failure.

HttpRequest::setQueryData

HttpRequest::setQueryData -- Set query data

Description

public bool **HttpRequest::setQueryData** (mixed `$query_data`)

Set the URL query parameters to use, overwriting previously set query parameters.

Affects any request types.

Parameters

`query_data`
: a string or associative array parameter containing the pre-encoded query string or to be encoded query fields; if empty, the query data will be unset

Return Values

Returns **TRUE** on success or **FALSE** on failure.

HttpRequest::setRawPostData

HttpRequest::setRawPostData -- Set raw post data

Description

public bool **HttpRequest::setRawPostData** ([string `$raw_post_data`])

Set raw post data to send, overwriting previously set raw post data. Don't forget to specify a content type. Affects only POST and custom requests.

Only either post fields or raw post data can be used for each request. Raw post data has higher precedence and will be used even if post fields are set.

Parameters

`raw_post_data`
raw post data

Return Values

Returns **TRUE** on success or **FALSE** on failure.

HttpRequest::setSslOptions

HttpRequest::setSslOptions -- Set ssl options

Description

public bool **HttpRequest::setSslOptions** ([array `$options`])

Set SSL options.

Parameters

`options`
: an associative array containing any SSL specific options; if empty or omitted, the SSL options will be reset

Return Values

Returns **TRUE** on success or **FALSE** on failure.

HttpRequest::setUrl

HttpRequest::setUrl -- Set URL

Description

public bool **HttpRequest::setUrl** (string `$url`)

Set the request URL.

Parameters

`url`
: the request url

Return Values

Returns **TRUE** on success or **FALSE** on failure.

The HttpRequestPool class

Class synopsis

HttpRequestPool

HttpRequestPool implements Iterator, Countable {

public bool **HttpRequestPool::attach** (HttpRequest `$request`)

void **HttpRequestPool::__construct** ([HttpRequest `$request`])

void **HttpRequestPool::__destruct** (void)

bool **HttpRequestPool::detach** (HttpRequest `$request`)

array **HttpRequestPool::getAttachedRequests** (void)

array **HttpRequestPool::getFinishedRequests** (void)

void **HttpRequestPool::reset** (void)

bool **HttpRequestPool::send** (void)

protected bool **HttpRequestPool::socketPerform** (void)

protected bool **HttpRequestPool::socketSelect** (void)
}

Class Members

Properties

The HttpRequestPool class does not have any properties.

Predefined Constants

The HttpRequestPool class does not have any constants.

HttpRequestPool::attach

HttpRequestPool::attach -- Attach HttpRequest

Description

public bool **HttpRequestPool::attach** (HttpRequest `$request`)

Attach an HttpRequest object to this HttpRequestPool.

Warning
Set all options prior attaching!

Parameters

`request`
: an HttpRequest object not already attached to any HttpRequestPool object

Return Values

Returns **TRUE** on success or **FALSE** on failure.

Errors/Exceptions

Throws HttpInvalidParamException, HttpRequestException, HttpRequestPoolException, HttpEncodingException.

HttpRequestPool::__construct

HttpRequestPool::__construct -- HttpRequestPool constructor

Description

void **HttpRequestPool::__construct** ([HttpRequest `$request`])

Instantiate a new HttpRequestPool object. An HttpRequestPool is able to send several HttpRequests in parallel.

Accepts virtually infinite optional parameters each referencing an HttpRequest object.

Parameters

`request`
HttpRequest object to attach

Errors/Exceptions

Throws HttpRequestPoolException (HttpRequestException, HttpInvalidParamException).

Examples

Example #6 - A HttpRequestPool example

```
<?php
try {
   $pool = new HttpRequestPool(
       new HttpRequest('http://www.google.com/', HttpRequest::METH_HEAD),
       new HttpRequest('http://www.php.net/', HttpRequest::METH_HEAD)
   );
   $pool->send();
   foreach($pool as $request) {
       printf("%s is %s (%d)\n",
           $request->getUrl(),
           $request->getResponseCode() ? 'alive' : 'not alive',
           $request->getResponseCode()
       );
   }
} catch (HttpException $e) {
   echo $e;
}
?>
```

HttpRequestPool::__destruct

HttpRequestPool::__destruct -- HttpRequestPool destructor

Description

void **HttpRequestPool::__destruct** (void)

Clean up HttpRequestPool object.

HttpRequestPool::detach

HttpRequestPool::detach -- Detach HttpRequest

Description

bool **HttpRequestPool::detach** (HttpRequest `$request`)

Detach an HttpRequest object from this HttpRequestPool.

Parameters

`request`
: an HttpRequest object attached to this HttpRequestPool object

Return Values

Returns **TRUE** on success or **FALSE** on failure.

Errors/Exceptions

Throws HttpInvalidParamException, HttpRequestPoolException.

HttpRequestPool::getAttachedRequests

HttpRequestPool::getAttachedRequests -- Get attached requests

Description

array **HttpRequestPool::getAttachedRequests** (void)

Get attached HttpRequest objects.

Parameters

Return Values

Returns an array containing all currently attached HttpRequest objects.

HttpRequestPool::getFinishedRequests

HttpRequestPool::getFinishedRequests -- Get finished requests

Description

array **HttpRequestPool::getFinishedRequests** (void)

Get attached HttpRequest objects that already have finished their work.

Parameters

Return Values

Returns an array containing all attached HttpRequest objects that already have finished their work.

HttpRequestPool::reset

HttpRequestPool::reset -- Reset request pool

Description

void **HttpRequestPool::reset** (void)

Detach all attached HttpRequest objects.

HttpRequestPool::send

HttpRequestPool::send -- Send all requests

Description

bool **HttpRequestPool::send** (void)

Send all attached HttpRequest objects in parallel.

Parameters

Return Values

Returns **TRUE** on success or **FALSE** on failure.

Errors/Exceptions

Throws HttpRequestPoolException (HttpSocketException, HttpRequestException, HttpMalformedHeaderException).

HttpRequestPool::socketPerform

HttpRequestPool::socketPerform -- Perform socket actions

Description

protected bool **HttpRequestPool::socketPerform** (void)

Returns TRUE until each request has finished its transaction.

Return Values

Returns TRUE until each request has finished its transaction.

Examples

Example #7 - A HttpRequestPool::socketPerform() example

```
<?php
class MyPool extends HttpRequestPool
{
   public function send()
   {
       while ($this->socketPerform()) {
           if (!$this->socketSelect()) {
               throw new HttpSocketExcpetion;
           }
       }
   }

   protected final function socketPerform()
   {
       $result = parent::socketPerform();
       foreach ($this->getFinishedRequests() as $r) {
           $this->detach($r);
           // handle response of finished request
       }
       return $result;
   }
}
?>
```

HttpRequestPool::socketSelect

HttpRequestPool::socketSelect -- Perform socket select

Description

protected bool **HttpRequestPool::socketSelect** (void)

Return Values

Returns **TRUE** on success or **FALSE** on failure.

The HttpResponse

Class synopsis

HttpResponse

HttpResponse {

static void **HttpResponse::capture** (void)

static int **HttpResponse::getBufferSize** (void)

static bool **HttpResponse::getCache** (void)

static string **HttpResponse::getCacheControl** (void)

static string **HttpResponse::getContentDisposition** (void)

static string **HttpResponse::getContentType** (void)

static string **HttpResponse::getData** (void)

static string **HttpResponse::getETag** (void)

static string **HttpResponse::getFile** (void)

static bool **HttpResponse::getGzip** (void)

static mixed **HttpResponse::getHeader** ([string `$name`])

static int **HttpResponse::getLastModified** (void)

static string **HttpResponse::getRequestBody** (void)

static resource **HttpResponse::getRequestBodyStream** (void)

static array **HttpResponse::getRequestHeaders** (void)

static resource **HttpResponse::getStream** (void)

static double **HttpResponse::getThrottleDelay** (void)

static string **HttpResponse::guessContentType** (string `$magic_file` [, int `$magic_mode=MAGIC_MIME`])

static void **HttpResponse::redirect** ([string `$url` [, array `$params` [, bool `$session = FALSE` [, int `$status`]]]])

static bool **HttpResponse::send** ([bool `$clean_ob = TRUE`])

static bool **HttpResponse::setBufferSize** (int `$bytes`)

static bool **HttpResponse::setCache** (bool `$cache`)

static bool **HttpResponse::setCacheControl** (string `$control` [, int `$max_age = 0` [, bool `$must_revalidate = TRUE`]])

static bool **HttpResponse::setContentDisposition** (string `$filename` [, bool `$inline = FALSE`])

static bool **HttpResponse::setContentType** (string `$content_type`)

static bool **HttpResponse::setData** (mixed `$data`)

static bool **HttpResponse::setETag** (string `$etag`)

static bool **HttpResponse::setFile** (string `$file`)

static bool **HttpResponse::setGzip** (bool `$gzip`)

static bool **HttpResponse::setHeader** (string `$name` [, mixed `$value` [, bool `$replace = TRUE`]])

static bool **HttpResponse::setLastModified** (int `$timestamp`)

static bool **HttpResponse::setStream** (resource `$stream`)

static bool **HttpResponse::setThrottleDelay** (float `$seconds`)

static bool **HttpResponse::status** (int `$status`)

}

Class Members

Properties

Static Properties

Modifiers	Type	Name	Description
protected	boolean	cache	whether caching the response should be attempted

protected	boolean	gzip	whether the sent entity should be gzip'ed on the fly
protected	string	eTag	the generated or custom ETag
protected	integer	lastModified	the generated or custom timestamp of last modification
protected	string	cacheControl	*Cache-Control* setting
protected	string	contentType	the *Content-Type* of the sent entity
protected	string	contentDisposition	the *Content-Disposition* of the sent entity
protected	integer	bufferSize	the chunk buffer size used for throttling
protected	double	throttleDelay	the seconds to delay when throttling

Predefined Constants

Type	**Name**	**Description**
integer	REDIRECT	guess applicable redirect method
integer	REDIRECT_PERM	permanent redirect (*301 Moved permanently*)
integer	REDIRECT_FOUND	standard redirect (*302 Found*)
integer	REDIRECT_POST	redirect applicable to POST requests (*303 See other*)
integer	REDIRECT_PROXY	proxy redirect (*305 Use proxy*)
integer	REDIRECT_TEMP	temporary redirect (*307 Temporary Redirect*)

HttpResponse::capture

HttpResponse::capture -- Capture script output

Description

static void **HttpResponse::capture** (void)

Capture script output.

Examples

Example #8 - A HttpResponse::capture() example

```
<?php
HttpResponse::setCache(true);
HttpResponse::capture();
// script follows
?>
```

See Also

- **HttpResponse::send**

HttpResponse::getBufferSize

HttpResponse::getBufferSize -- Get buffer size

Description

static int **HttpResponse::getBufferSize** (void)

Get current buffer size.

Return Values

Returns an int representing the current buffer size in bytes.

See Also

- **HttpResponse::setBufferSize**
- **HttpResponse::getThrottleDelay**
- **HttpResponse::setThrottleDelay**

HttpResponse::getCache

HttpResponse::getCache -- Get cache

Description

static bool **HttpResponse::getCache** (void)

Get current caching setting.

Return Values

Returns **TRUE** if caching should be attempted, else **FALSE**.

See Also

- **HttpResponse::setCacheControl**
- **HttpResponse::getCacheControl**
- **HttpResponse::setCache**

HttpResponse::getCacheControl

HttpResponse::getCacheControl -- Get cache control

Description

static string **HttpResponse::getCacheControl** (void)

Get current *Cache-Control* header setting.

Return Values

Returns the current cache control setting as a string like sent in a header.

See Also

- **HttpResponse::setCacheControl**
- **HttpResponse::setCache**
- **HttpResponse::getCachel**

HttpResponse::getContentDisposition

HttpResponse::getContentDisposition -- Get content disposition

Description

static string **HttpResponse::getContentDisposition** (void)

Get current *Content-Disposition* setting.

Return Values

Returns the current content disposition as string like sent in a header.

See Also

- **HttpResponse::setContentDisposition**
- **HttpResponse::getContentType**
- **HttpResponse::setContentType**

HttpResponse::getContentType

HttpResponse::getContentType -- Get content type

Description

static string **HttpResponse::getContentType** (void)

Get current *Content-Type* header setting.

Return Values

Returns the currently set content type as string.

See Also

- **HttpResponse::getContentDisposition**
- **HttpResponse::setContentDisposition**
- **HttpResponse::setContentType**
- **HttpResponse::guessContentType**

HttpResponse::getData

HttpResponse::getData -- Get data

Description

static string **HttpResponse::getData** (void)

Get the previously set data to be sent.

Return Values

Returns a string containing the previously set data to send.

See Also

- **HttpResponse::setData**
- **HttpResponse::getFile**
- **HttpResponse::setFile**
- **HttpResponse::getStream**
- **HttpResponse::setStream**

HttpResponse::getETag

HttpResponse::getETag -- Get ETag

Description

static string **HttpResponse::getETag** (void)

Get calculated or previously set custom *ETag*.

Return Values

Returns the calculated or previously set *ETag* as unquoted string.

See Also

- **HttpResponse::getLastModified**
- **HttpResponse::setLastModified**
- **HttpResponse::setETag**

HttpResponse::getFile

HttpResponse::getFile -- Get file

Description

static string **HttpResponse::getFile** (void)

Get the previously set file to be sent.

Return Values

Returns the previously set path to the file to send as string.

See Also

- **HttpResponse::getData**
- **HttpResponse::setData**
- **HttpResponse::setFile**
- **HttpResponse::getStream**
- **HttpResponse::setStream**

HttpResponse::getGzip

HttpResponse::getGzip -- Get gzip

Description

static bool **HttpResponse::getGzip** (void)

Get current gzip'ing setting.

Return Values

Returns **TRUE** if GZip compression is enabled, else **FALSE**.

See Also

- **HttpResponse::setGzip**

HttpResponse::getHeader

HttpResponse::getHeader -- Get header

Description

static mixed **HttpResponse::getHeader** ([string `$name`])

Get header(s) about to be sent.

Note
This may not work as expected with the following SAPI(s): Apache2 w/PHP < 5.1.3.

Parameters

`name`
: specifies the name of the header to read; if empty or omitted, an associative array with all headers will be returned

Return Values

Returns either a string containing the value of the header matching name, **FALSE** on failure, or an associative array with all headers.

See Also

- **HttpResponse::setHeader**

HttpResponse::getLastModified

HttpResponse::getLastModified -- Get last modified

Description

static int **HttpResponse::getLastModified** (void)

Get calculated or previously set custom *Last-Modified* date.

Return Values

Returns the calculated or previously set Unix timestamp.

See Also

- **HttpResponse::setLastModified**
- **HttpResponse::getETag**
- **HttpResponse::setETag**

HttpResponse::getRequestBody

HttpResponse::getRequestBody -- Get request body

Description

static string **HttpResponse::getRequestBody** (void)

This function is an alias of: http_get_request_body().

HttpResponse::getRequestBodyStream

HttpResponse::getRequestBodyStream -- Get request body stream

Description

static resource **HttpResponse::getRequestBodyStream** (void)

This function is an alias of: http_get_request_body_stream().

HttpResponse::getRequestHeaders

HttpResponse::getRequestHeaders -- Get request headers

Description

static array **HttpResponse::getRequestHeaders** (void)

This function is an alias of: http_get_request_headers().

HttpResponse::getStream

HttpResponse::getStream -- Get Stream

Description

static resource **HttpResponse::getStream** (void)

Get the previously set resource to be sent.

Parameters

Return Values

Returns the previously set resource.

See Also

- **HttpResponse::getData**
- **HttpResponse::setData**
- **HttpResponse::getFile**
- **HttpResponse::setFile**
- **HttpResponse::setStream**

HttpResponse::getThrottleDelay

HttpResponse::getThrottleDelay -- Get throttle delay

Description

static double **HttpResponse::getThrottleDelay** (void)

Get the current throttle delay.

Return Values

Returns a double representing the throttle delay in seconds.

See Also

- **HttpResponse::getBufferSize**
- **HttpResponse::setBufferSize**
- **HttpResponse::setThrottleDelay**

HttpResponse::guessContentType

HttpResponse::guessContentType -- Guess content type

Description

static string **HttpResponse::guessContentType** (string `$magic_file` [, int `$magic_mode=MAGIC_MIME`])

Attempts to guess the content type of supplied payload through libmagic.

If the attempt is successful, the guessed *Content-Type* will automatically be set as response *Content-Type*.

Parameters

`magic_file`
: specifies the magic.mime database to use

`magic_mode`
: flags for libmagic

Return Values

Returns the guessed content type on success, or **FALSE** on failure.

Errors/Exceptions

Throws HttpRuntimeException, HttpInvalidParamException.

See Also

- **HttpResponse::getContentType**
- **HttpResponse::setContentType**

HttpResponse::redirect

HttpResponse::redirect -- Redirect

Description

static void **HttpResponse::redirect** ([string `$url` [, array `$params` [, bool `$session = FALSE` [, int `$status`]]]])

This function is an alias of: http_redirect().

HttpResponse::send

HttpResponse::send -- Send response

Description

static bool **HttpResponse::send** ([bool `$clean_ob = TRUE`])

Finally send the entity.

A successful caching attempt will exit PHP, and write a log entry if the INI setting http.log.cache is set. See the INI setting http.force_exit for what "exits" means.

Parameters

`clean_ob`
: whether to destroy all previously started output handlers and their buffers

Return Values

Returns **TRUE** on success or **FALSE** on failure.

Examples

Example #9 - A HttpResponse::send() example

```
<?php
HttpResponse::setCache(true);
HttpResponse::setContentType('application/pdf');
HttpResponse::setContentDisposition("$user.pdf", false);
HttpResponse::setFile('sheet.pdf');
HttpResponse::send();
?>
```

See Also

- **HttpResponse::capture**

HttpResponse::setBufferSize

HttpResponse::setBufferSize -- Set buffer size

Description

static bool **HttpResponse::setBufferSize** (int `$bytes`)

Sets the send buffer size of the throttling mechanism.

Note
Provides a basic throttling mechanism, which will yield the current process or thread until the entity has been completely sent.

Note
This may not work as expected with the following SAPI(s): FastCGI.

Parameters

`bytes`
: the chunk size in bytes

Return Values

Returns **TRUE** on success or **FALSE** on failure.

See Also

- **HttpResponse::getBufferSize**
- **HttpResponse::getThrottleDelay**
- **HttpResponse::setThrottleDelay**

HttpResponse::setCache

HttpResponse::setCache -- Set cache

Description

static bool **HttpResponse::setCache** (bool `$cache`)

Whether it should be attempted to cache the entity.

This will result in necessary caching headers and checks of clients *If-Modified-Since* and *If-None-Match* headers. If one of those headers matches a *304 Not Modified* status code will be issued.

Note
If you're using sessions, be sure that you set session.cache_limiter to something more appropriate than "no-cache"!

Parameters

cache
: whether caching should be attempted

Return Values

Returns **TRUE** on success or **FALSE** on failure.

See Also

- **HttpResponse::setCacheControl**
- **HttpResponse::getCacheControl**
- **HttpResponse::getCachel**

HttpResponse::setCacheControl

HttpResponse::setCacheControl -- Set cache control

Description

static bool **HttpResponse::setCacheControl** (string `$control` [, int `$max_age = 0` [, bool `$must_revalidate = TRUE`]])

Define a custom *Cache-Control* header, usually being *private* or *public*;

Parameters

`control`
: the primary cache control setting

`max_age`
: the max-age in seconds, suggesting how long the cache entry is valid on the client side

`must_revalidate`
: whether the cached entity should be revalidated by the client for every request

Return Values

Returns **TRUE** on success, or **FALSE** if control does not match one of *public*, *private* or *no-cache*.

See Also

- HttpResponse::getCacheControl()
- HttpResponse::setCache()
- HttpResponse::getCache()

HttpResponse::setContentDisposition

HttpResponse::setContentDisposition -- Set content disposition

Description

static bool **HttpResponse::setContentDisposition** (string `$filename` [, bool `$inline = FALSE`])

Set the *Content-Disposition*. The *Content-Disposition* header is very useful if the data actually being sent came from a file or something similar, that should be "saved" by the client/user (i.e. by the browser's "Save as..." popup window).

Parameters

`filename`
: the file name the "Save as..." dialog should display

`inline`
: if set to true and the user agent knows how to handle the content type, it will probably not cause the popup window to be shown

Return Values

Returns **TRUE** on success or **FALSE** on failure.

See Also

- **HttpResponse::getContentDisposition**
- **HttpResponse::getContentType**
- **HttpResponse::setContentType**

HttpResponse::setContentType

HttpResponse::setContentType -- Set content type

Description

static bool **HttpResponse::setContentType** (string `$content_type`)

Set the *Content-Type* of the sent entity.

Parameters

`content_type`
: the content type of the sent entity (primary/secondary)

Return Values

Returns **TRUE** on success, or **FALSE** if the content type does not seem to contain a primary and secondary content type part.

See Also

- **HttpResponse::getContentDisposition**
- **HttpResponse::setContentDisposition**
- **HttpResponse::getContentType**
- **HttpResponse::guessContentType**

HttpResponse::setData

HttpResponse::setData -- Set data

Description

static bool **HttpResponse::setData** (mixed `$data`)

Set the data to be sent.

Note
Previously calculated or defined *ETag* and *Last-Modified* will be recalculated and redefined.

Parameters

`data`
: data to send

Return Values

Returns **TRUE** on success or **FALSE** on failure.

See Also

- **HttpResponse::getData**
- **HttpResponse::getFile**
- **HttpResponse::setFile**
- **HttpResponse::getStream**
- **HttpResponse::setStream**

HttpResponse::setETag

HttpResponse::setETag -- Set ETag

Description

static bool **HttpResponse::setETag** (string `$etag`)

Set a custom *ETag*. Use this only if you know what you're doing.

Parameters

`etag`
: unquoted string as parameter containing the ETag

Return Values

Returns **TRUE** on success or **FALSE** on failure.

See Also

- **HttpResponse::getLastModified**
- **HttpResponse::setLastModified**
- **HttpResponse::getETag**

HttpResponse::setFile

HttpResponse::setFile -- Set file

Description

static bool **HttpResponse::setFile** (string `$file`)

Set the file to be sent.

Note
Previously calculated or defined *ETag* and *Last-Modified* will be recalculated and redefined.

Parameters

`file`
: the path to the file to send

Return Values

Returns **TRUE** on success or **FALSE** on failure.

See Also

- **HttpResponse::getData**
- **HttpResponse::setData**
- **HttpResponse::getFile**
- **HttpResponse::getStream**
- **HttpResponse::setStream**

HttpResponse::setGzip

HttpResponse::setGzip -- Set gzip

Description

static bool **HttpResponse::setGzip** (bool `$gzip`)

Enable on-thy-fly gzip'ing of the sent entity.

Parameters

`gzip`
: whether GZip compression should be enabled

Return Values

Returns **TRUE** on success or **FALSE** on failure.

See Also

- **HttpResponse::getGzip**

HttpResponse::setHeader

HttpResponse::setHeader -- Set header

Description

static bool **HttpResponse::setHeader** (string `$name` [, mixed `$value` [, bool `$replace = TRUE`]])

Send an HTTP header.

Parameters

`name`
: the name of the header

`value`
: the value of the header; if not set, no header with this name will be sent

`replace`
: whether an existing header should be replaced

Return Values

Returns **TRUE** on success or **FALSE** on failure.

See Also

- **HttpResponse::getHeader**

HttpResponse::setLastModified

HttpResponse::setLastModified -- Set last modified

Description

static bool **HttpResponse::setLastModified** (int `$timestamp`)

Set a custom *Last-Modified* date.

Parameters

`timestamp`
: Unix timestamp representing the last modification time of the sent entity

Return Values

Returns **TRUE** on success or **FALSE** on failure.

See Also

- **HttpResponse::getLastModified**
- **HttpResponse::getETag**
- **HttpResponse::setETag**

HttpResponse::setStream

HttpResponse::setStream -- Set stream

Description

static bool **HttpResponse::setStream** (resource `$stream`)

Set the resource to be sent.

Note
Previously calculated or defined *ETag* and *Last-Modified* will be recalculated and redefined.

Parameters

`stream`
: already opened stream from which the data to send will be read

Return Values

Returns **TRUE** on success or **FALSE** on failure.

See Also

- **HttpResponse::getData**
- **HttpResponse::setData**
- **HttpResponse::getFile**
- **HttpResponse::setFile**
- **HttpResponse::getStream**

HttpResponse::setThrottleDelay

HttpResponse::setThrottleDelay -- Set throttle delay

Description

static bool **HttpResponse::setThrottleDelay** (float `$seconds`)

Sets the throttle delay.

Note
Provides a basic throttling mechanism, which will yield the current process or thread until the entity has been completely sent.

Note
This may not work as expected with the following SAPI(s): FastCGI.

Parameters

seconds
: seconds to sleep after each chunk sent

Return Values

Returns **TRUE** on success or **FALSE** on failure.

See Also

- **HttpResponse::getBufferSize**
- **HttpResponse::setBufferSize**
- **HttpResponse::getThrottleDelay**

HttpResponse::status

HttpResponse::status -- Send HTTP response status

Description

static bool **HttpResponse::status** (int `$status`)

This function is an alias of: http_send_status().

HTTP Functions

Built-in HTTP related functions previously listed on this page can be found in the networking category.

The following functions do not need the HTTP module to be present: header(), headers_list(), headers_sent(), setcookie() and setrawcookie().

Function groups

Caching

- http_cache_etag()
- http_cache_last_modified()

Encodings

- http_chunked_decode()
- http_deflate()
- http_inflate()

Miscellaneous

- http_build_cookie()
- http_date()
- http_get_request_body_stream()
- http_get_request_body()
- http_get_request_headers()
- http_match_etag()
- http_match_modified()
- http_match_request_header()
- http_support()

Negotiation

- http_negotiate_charset()
- http_negotiate_content_type()
- http_negotiate_language()

Outputhandlers

- ob_deflatehandler()
- ob_etaghandler()
- ob_inflatehandler()

Parsers

- http_parse_cookie()
- http_parse_headers()
- http_parse_message()
- http_parse_params()

Requests

- http_get()
- http_head()
- http_post_data()
- http_post_fields()
- http_put_data()
- http_put_file()
- http_put_stream()
- http_request_body_encode()
- http_request_method_exists()
- http_request_method_name()
- http_request_method_register()
- http_request_method_unregister()
- http_request()

Responses

- http_redirect()
- http_send_content_disposition()
- http_send_content_type()
- http_send_data()
- http_send_file()
- http_send_last_modified()
- http_send_status()
- http_send_stream()
- http_throttle()

URLs

- http_build_str()
- http_build_url()

Persistent Handles

- http_persistent_handles_count()
- http_persistent_handles_ident()
- http_persistent_handles_clean()

http_cache_etag

http_cache_etag -- Caching by ETag

Description

bool **http_cache_etag** ([string `$etag`])

Attempts to cache the sent entity by its *ETag*, either supplied or generated by the hash algorithm specified by the INI setting http.etag.mode.

If the clients *If-None-Match* header matches the supplied/calculated ETag, the body is considered cached on the clients side and a *304 Not Modified* status code is issued.

A log entry is written to the cache log if the INI setting http.log.cache is set and the cache attempt was successful.

Note
This function may be used in conjunction with http_send_data(), http_send_file() and http_send_stream().

If this function is used outside the http_send_*() API, it facilitates the ob_etaghandler().

Parameters

`etag`
custom *ETag*

Return Values

Returns **FALSE** or *exits* on success with *304 Not Modified* if the entity is cached. See the INI setting http.force_exit for what "exits" means.

Examples

Example #10 - A http_cache_etag() example

```
<?php
http_cache_etag();
http_send_data("data");
?>
```

See Also

- http_cache_last_modified()
- ob_etaghandler()
- http_match_etag()
- the HttpResponse class if you are using PHP 5.1.0 and above

http_cache_last_modified

http_cache_last_modified -- Caching by last modification

Description

bool **http_cache_last_modified** ([int `$timestamp_or_expires`])

Attempts to cache the sent entity by its last modification date.

If the supplied argument is greater than 0, it is handled as timestamp and will be sent as date of last modification. If it is 0 or omitted, the current time will be sent as *Last-Modified* date. If it's negative, it is handled as expiration time in seconds, which means that if the requested last modification date is not between the calculated timespan, the *Last-Modified* header is updated and the actual body will be sent.

A log entry will be written to the cache log if the INI setting http.log.cache is set and the cache attempt was successful.

Note
This function may be used in conjunction with http_send_data(), http_send_file() and http_send_stream().

Parameters

`timestamp_or_expires`
: Unix timestamp

Return Values

Returns **FALSE** or *exits* on success with *304 Not Modified* if the entity is cached. See the INI setting http.force_exit for what "exits" means.

Examples

Example #11 - A http_cache_last_modified() example

Caching for 5 seconds.

```
<?php
http_cache_last_modified(-5);
printf("%s\n", http_date());
```

```
?>
```

See Also

- http_cache_etag()
- the HttpResponse class if you are using PHP 5.1.0 and above

http_chunked_decode

http_chunked_decode -- Decode chunked-encoded data

Description

string **http_chunked_decode** (string `$encoded`)

Decodes a string which is HTTP-chunked encoded.

Parameters

encoded
chunked encoded string

Return Values

Returns the decoded string on success or **FALSE** on failure.

Examples

Example #12 - A http_chunked_decode() example

```
<?php
$string = "".
   "05\r\n".
   "this \r\n".
   "07\r\n".
   "string \r\n".
   "12\r\n".
   "is chunked encoded\r\n".
   "01\n\r\n".
   "00";
echo http_chunked_decode($string);
?>
```

The above example will output:

```
this string is chunked encoded
```

http_deflate

http_deflate -- Deflate data

Description

string **http_deflate** (string `$data` [, int `$flags = 0`])

Compress data with *gzip*, *zlib* AKA *deflate* or *raw deflate* encoding.

See the deflate constants table for possible values for the *flags* parameter.

Parameters

`data`
: String containing the data that should be encoded

`flags`
: deflate options

Return Values

Returns the encoded string on success, or NULL on failure.

See Also

- http_inflate()
- HttpDeflateStream

http_inflate

http_inflate -- Inflate data

Description

string **http_inflate** (string `$data`)

Decompress data compressed with either *gzip*, *deflate* AKA *zlib* or *raw deflate* encoding.

Parameters

`data`
: string containing the compressed data

Return Values

Returns the decoded string on success, or NULL on failure.

See Also

- http_deflate()
- HttpInflateStream
-

http_build_cookie

http_build_cookie -- Build cookie string

Description

string **http_build_cookie** (array `$cookie`)

Build a cookie string from an array/object like returned by http_parse_cookie().

Parameters

`cookie`
: a cookie list like returned from http_parse_cookie()

Return Values

Returns the cookie(s) as string.

See Also

- http_parse_cookie()

http_date

http_date -- Compose HTTP RFC compliant date

Description

string **http_date** ([int `$timestamp`])

Compose a valid HTTP date regarding RFC 1123 looking like: *Wed, 22 Dec 2004 11:34:47 GMT*.

Parameters

`timestamp`
: Unix timestamp; current time if omitted

Return Values

Returns the HTTP date as string.

See Also

- date()

http_get_request_body_stream

http_get_request_body_stream -- Get request body as stream

Description

resource **http_get_request_body_stream** (void)

Create a stream to read the raw request body (e.g. POST or PUT data).

This function can only be used once if the request method was another than POST.

Parameters

Return Values

Returns the raw request body as stream on success or NULL on failure.

See Also

- http_get_request_body()
- http_get_request_headers()
- the HttpResponse class if you are using PHP 5.1.0 and above

http_get_request_body

http_get_request_body -- Get request body as string

Description

string **http_get_request_body** (void)

Get the raw request body (e.g. POST or PUT data).

This function can not be used after http_get_request_body_stream() if the request method was another than POST.

Parameters

Return Values

Returns the raw request body as string on success or NULL on failure.

See Also

- http_get_request_body_stream()
- http_get_request_headers()
- the HttpResponse class if you are using PHP 5.1.0 and above

http_get_request_headers

http_get_request_headers -- Get request headers as array

Description

array **http_get_request_headers** (void)

Get a list of incoming HTTP headers.

Parameters

Return Values

Returns an associative array of incoming request headers.

See Also

- http_get_request_body()
- http_get_request_body_stream()
- the HttpResponse class if you are using PHP 5.1.0 and above

http_match_etag

http_match_etag -- Match ETag

Description

bool **http_match_etag** (string `$etag` [, bool `$for_range = FALSE`])

Matches the given *ETag* against the clients *If-Match* resp. *If-None-Match* HTTP headers.

Parameters

`etag`
: the *ETag* to match

`for_range`
: if set to **TRUE**, the header usually used to validate HTTP ranges will be checked

Return Values

Returns **TRUE** if *ETag* matches or the header contained the asterisk ("*"), else **FALSE**.

See Also

- **http_match_last_modified()**
- http_match_request_header()
- http_cache_etag()
- http_cache_last_modified()
- ob_etaghandler()

http_match_modified

http_match_modified -- Match last modification

Description

bool **http_match_modified** ([int `$timestamp` [, bool `$for_range = FALSE`]])

Matches the given Unix timestamp against the clients *If-Modified-Since* resp. *If-Unmodified-Since* HTTP headers.

Parameters

`timestamp`
: Unix timestamp; current time, if omitted

`for_range`
: if set to **TRUE**, the header usually used to validate HTTP ranges will be checked

Return Values

Returns **TRUE** if timestamp represents an earlier date than the header, else **FALSE**.

See Also

- http_match_etag()
- http_match_request_header()
- http_cache_etag()
- http_cache_last_modified()

http_match_request_header

http_match_request_header -- Match any header

Description

bool **http_match_request_header** (string `$header`, string `$value` [, bool `$match_case = FALSE`])

Match an incoming HTTP header.

Parameters

`header`
: the header name (case-insensitive)

`value`
: the header value that should be compared

`match_case`
: whether the value should be compared case sensitively

Return Values

Returns **TRUE** if header value matches, else **FALSE**.

See Also

- http_match_etag()
- **http_match_last_modified()**

http_support

http_support -- Check built-in HTTP support

Description

int **http_support** ([int `$feature = 0`])

Check for features that require external libraries.

See the feature support constants table for possible values for the *`feature`* argument.

Parameters

`feature`
: feature to probe for

Return Values

Returns integer, whether requested feature is supported, or a bitmask with all supported features if *`feature`* was omitted.

Examples

Example #13 - A http_support() example

```
<?php
if (!http_support(HTTP_SUPPORT_REQUESTS)) {
   die("Need HTTP request support!\n");
}
?>
```

http_negotiate_charset

http_negotiate_charset -- Negotiate clients preferred character set

Description

string **http_negotiate_charset** (array `$supported` [, array `&$result`])

This function negotiates the clients preferred charset based on its *Accept-Charset* HTTP header. The qualifier is recognized and charsets without qualifier are rated highest.

Parameters

`supported`
array containing the supported charsets as values

`result`
will be filled with an array containing the negotiation results

Return Values

Returns the negotiated charset or the default charset (i.e. first array entry) if none match.

Examples

Example #14 - Using http_negotiate_charset()

```
<?php
$charsets = array(
       'iso-8859-1', // default
       'iso-8859-2',
       'iso-8859-15',
       'utf-8'
);

$pref = http_negotiate_charset($charsets, $result);

if (strcmp($pref, 'iso-8859-1')) {
       iconv_set_encoding('internal_encoding', 'iso-8859-1');
       iconv_set_encoding('output_encoding', $pref);
       ob_start('ob_iconv_handler');
}

print_r($result);
?>
```

http_negotiate_content_type

http_negotiate_content_type -- Negotiate clients preferred content type

Description

string **http_negotiate_content_type** (array `$supported` [, array `&$result`])

This function negotiates the clients preferred content type based on its *Accept* HTTP header. The qualifier is recognized and content types without qualifier are rated highest.

Parameters

`supported`
: array containing the supported content types as values

`result`
: will be filled with an array containing the negotiation results

Return Values

Returns the negotiated content type or the default content type (i.e. first array entry) if none match.

Examples

Example #15 - Using http_negotiate_content_type()

```
<?php
$content_types = array('application/xhtml+xml', 'text/html');
http_send_content_type(http_negotiate_content_type($content_types));
?>
```

http_negotiate_language

http_negotiate_language -- Negotiate clients preferred language

Description

string **http_negotiate_language** (array `$supported` [, array `&$result`])

This function negotiates the clients preferred language based on its *Accept-Language* HTTP header. The qualifier is recognized and languages without qualifier are rated highest. The qualifier will be decreased by 10% for partial matches (i.e. matching primary language).

Parameters

`supported`
array containing the supported languages as values

`result`
will be filled with an array containing the negotiation results

Return Values

Returns the negotiated language or the default language (i.e. first array entry) if none match.

Examples

Example #16 - Using http_negotiate_language()

```
<?php
$langs = array(
       'en-US',// default
       'fr',
       'fr-FR',
       'de',
       'de-DE',
       'de-AT',
       'de-CH',
);

include './langs/'. http_negotiate_language($langs, $result) .'.php';

print_r($result);
?>
```

ob_deflatehandler

ob_deflatehandler -- Deflate output handler

Description

string **ob_deflatehandler** (string `$data`, int `$mode`)

For use with ob_start().

Note
This output handler can only be used once.

The deflate output buffer handler can only be used once.

It conflicts with ob_gzhandler() and zlib.output_compression as well and should not be used after mbstring extension's mb_output_handler() and session extension's URL-Rewriter (AKA session.use_trans_sid).

See Also

- ob_inflatehandler()
- ob_start()

ob_etaghandler

ob_etaghandler -- ETag output handler

Description

string **ob_etaghandler** (string `$data`, int `$mode`)

For use with ob_start().

Output buffer handler generating an *ETag* with the hash algorithm specified with the INI setting http.etag.mode.

This output handler is used by http_cache_etag().

See Also

- http_cache_etag()
- http_match_etag()

ob_inflatehandler

ob_inflatehandler -- Inflate output handler

Description

string **ob_inflatehandler** (string `$data`, int `$mode`)

For use with ob_start().

Same restrictions as with ob_deflatehandler() apply.

See Also

- ob_deflatehandler()
- ob_start()

http_parse_cookie

http_parse_cookie -- Parse HTTP cookie

Description

object **http_parse_cookie** (string `$cookie` [, int `$flags` [, array `$allowed_extras`]])

Parses HTTP cookies like sent in a response into a struct.

Parameters

`cookie`
: string containing the value of a *Set-Cookie* response header

`flags`
: parse flags (**HTTP_COOKIE_PARSE_RAW**)

`allowed_extras`
: array containing recognized *extra* keys; by default all unknown keys will be treated as cookie names

Return Values

Returns a stdClass object on success or **FALSE** on failure.

Examples

Example #17 - Using http_parse_cookie()

```
<?php
print_r(http_parse_cookie("foo=bar; bar=baz; path=/; domain=example.com;
comment=; secure", 0, array("comment")));
?>
```

The above example will output:

```
stdClass Object
(
 [cookies] => Array
 (
   [foo] => bar
   [bar] => baz
 )

 [extras] => Array
 (
```

```
    [comment] =>
  )

  [flags] => 16
  [expires] => 0
  [path] => /
  [domain] => example.com
 )
```

See Also

- http_parse_headers()
- http_parse_message()
- http_build_cookie()

http_parse_headers

http_parse_headers -- Parse HTTP headers

Description

array **http_parse_headers** (string `$header`)

Parses HTTP headers into an associative array.

Parameters

`header`
string containing HTTP headers

Return Values

Returns an array on success, or **FALSE** on failure.

Examples

Example #18 - Using http_parse_headers()

```
<?php
$headers = "content-type: text/html; charset=UTF-8\r\n".
 "Server: Funky/1.0\r\n".
 "Set-Cookie: foo=bar\r\n".
 "Set-Cookie: baz=quux\r\n".
 "Folded: works\r\n\ttoo\r\n";
print_r(http_parse_headers($headers));
?>
```

The above example will output:

```
Array
(
 [Content-Type] => text/html; chatset=UTF-8
 [Server] => Funky/1.0
 [Set-Cookie] => Array
 (
   [0] => foo=bar
   [1] => baz=quux
 )
 [Folded] => works
   too
)
```

See Also

- http_parse_message()
- http_parse_cookie()

http_parse_message

http_parse_message -- Parse HTTP messages

Description

object **http_parse_message** (string `$message`)

Parses the HTTP *`message`* into a simple recursive object.

Parameters

`message`
string containing a single HTTP message or several consecutive HTTP messages

Return Values

Returns a hierarchical object structure of the parsed messages.

Examples

Example #19 - Using http_parse_message()

```
<?php
define ('URL', 'http://www.example.com/');
print_r(http_parse_message(http_get(URL, array('redirect' => 3))));
?>
```

The above example will output something similar to:

```
stdClass object
(
 [type] => 2
 [httpVersion] => 1.1
 [responseCode] => 200
 [headers] => Array
 (
   [Content-Length] => 3
   [Server] => Apache
 )
 [body] => Hi!
 [parentMessage] => stdClass object
 (
   [type] => 2
   [httpVersion] => 1.1
   [responseCode] => 302
   [headers] => Array
   (
```

```
      [Content-Length] => 0
      [Location] => ...
    )
    [body] =>
    [parentMessage] => ...
  )
)
```

See Also

- http_parse_headers()
- the HttpMessage class

http_parse_params

http_parse_params -- Parse parameter list

Description

object **http_parse_params** (string `$param` [, int `$flags = HTTP_PARAMS_DEFAULT`])

Parse parameter list.

See the params parsing constants table for possible values of the *`flags`* argument.

Parameters

`param`
Parameters

`flags`
Parse flags

Return Values

Returns parameter list as stdClass object.

Examples

Example #20 - A http_parse_params() example

```
<?php
var_dump(http_parse_params("text/html; charset=\"utf8\""));
?>
```

The above example will output:

```
object(stdClass)#1 (1) {
 ["params"]=>
 array(2) {
   [0]=>
   string(9) "text/html"
   [1]=>
   array(1) {
     ["charset"]=>
     string(4) "utf8"
   }
 }
}
```

See Also

- http_parse_headers()
- http_parse_cookie()
- http_parse_message()

http_persistent_handles_clean

http_persistent_handles_clean -- Clean up persistent handles

Description

string **http_persistent_handles_clean** ([string `$ident`])

Clean up (close) persistent handles, optionally identified with ident.

Parameters

`clean`
: the identification string

Return Values

No value is returned.

See Also

- http_persistent_handles_ident()
- http_persistent_handles_count()

http_persistent_handles_count

http_persistent_handles_count -- Stat persistent handles

Description

object **http_persistent_handles_count** (void)

List statistics about persistent handles usage.

Parameters

Return Values

Returns persistent handles statistics as stdClass object on success or **FALSE** on failure.

Examples

Example #21 - A http_persistent_handles_count() example

```
<?php
print_r(http_persistent_handles_count());
?>
```

The above example will output:

```
stdClass Object
(
   [http_request] => Array
       (
           [GLOBAL] => Array
               (
                   [used] => 0
                   [free] => 1
               )

       )

   [http_request_datashare] => Array
       (
           [GLOBAL] => Array
               (
                   [used] => 1
                   [free] => 0
               )

       )
```

```
    [http_request_pool] => Array
        (
        )

)
```

See Also

- http_persistent_handles_ident()
- http_persistent_handles_clean()

http_persistent_handles_ident

http_persistent_handles_ident -- Get/set ident of persistent handles

Description

string **http_persistent_handles_ident** (string `$ident`)

Query or define the ident of persistent handles.

Parameters

`ident`
the identification string

Return Values

Returns the prior ident as string on success or **FALSE** on failure.

Examples

Example #22 - A http_persistent_handles_ident() example

```
<?php
echo http_persistent_handles_ident("CUSTOM"), "\n";
echo http_persistent_handles_ident("MyApp1"), "\n";
http_get("http://www.example.com/");
print_r(http_persistent_handles_count());
?>
```

The above example will output:

```
GLOBAL
CUSTOM
stdClass Object
(
   [http_request] => Array
       (
           [MyApp1] => Array
               (
                   [used] => 0
                   [free] => 1
               )

       )

   [http_request_datashare] => Array
```

```
        (
            [GLOBAL] => Array
                (
                    [used] => 1
                    [free] => 0
                )

        )

    [http_request_pool] => Array
        (
        )

)
```

See Also

- http_persistent_handles_count()
- http_persistent_handles_clean()

http_get

http_get -- Perform GET request

Description

string **http_get** (string `$url` [, array `$options` [, array `&$info`]])

Performs an HTTP GET request on the supplied url.

See the full list of request options.

Parameters

`url`
URL

`options`
request options

`info`
Will be filled with request/response information

Return Values

Returns the HTTP response(s) as string on success, or **FALSE** on failure.

Examples

Example #23 - A http_get() example

```
<?php
$response = http_get("http://www.example.com/", array("timeout"=>1), $info);
print_r($info);
?>
```

The above example will output:

```
array (
 'effective_url' => 'http://www.example.com/',
 'response_code' => 302,
 'connect_code' => 0,
 'filetime' => -1,
 'total_time' => 0.212348,
 'namelookup_time' => 0.038296,
 'connect_time' => 0.104144,
```

```
 'pretransfer_time' => 0.104307,
 'starttransfer_time' => 0.212077,
 'redirect_time' => 0,
 'redirect_count' => 0,
 'size_upload' => 0,
 'size_download' => 218,
 'speed_download' => 1026,
 'speed_upload' => 0,
 'header_size' => 307,
 'request_size' => 103,
 'ssl_verifyresult' => 0,
 'ssl_engines' =>
 array (
   0 => 'dynamic',
   1 => 'cswift',
   2 => 'chil',
   3 => 'atalla',
   4 => 'nuron',
   5 => 'ubsec',
   6 => 'aep',
   7 => 'sureware',
   8 => '4758cca',
 ),
 'content_length_download' => 218,
 'content_length_upload' => 0,
 'content_type' => 'text/html',
 'httpauth_avail' => 0,
 'proxyauth_avail' => 0,
 'num_connects' => 1,
 'os_errno' => 0,
 'error' => '',
)
```

http_head

http_head -- Perform HEAD request

Description

string **http_head** ([string `$url` [, array `$options` [, array `&$info`]]])

Performs an HTTP HEAD request on the supplied url.

See the full list of request options.

Parameters

`url`
: URL

`options`
: request options

`info`
: Request/response information

Return Values

Returns the HTTP response(s) as string on success, or **FALSE** on failure.

http_post_data

http_post_data -- Perform POST request with pre-encoded data

Description

string **http_post_data** (string `$url` [, string `$data` [, array `$options` [, array `&$info`]]])

Performs an HTTP POST request on the supplied url.

See the full list of request options.

Parameters

`url`
: URL

`data`
: String containing the pre-encoded post data

`options`
: request options

`info`
: Request/response information

Return Values

Returns the HTTP response(s) as string on success, or **FALSE** on failure.

http_post_fields

http_post_fields -- Perform POST request with data to be encoded

Description

string **http_post_fields** (string `$url` [, array `$data` [, array `$files` [, array `$options` [, array `&$info`]]]])

Performs an HTTP POST request on the supplied url.

See the full list of request options.

Parameters

`url`
URL

`data`
Associative array of POST values

`files`
Array of files to post

`options`
request options

`info`
Request/response information

Return Values

Returns the HTTP response(s) as string on success, or **FALSE** on failure.

Examples

Example #24 - A http_post_fields() example

```
<?php
$fields = array(
   'name' => 'mike',
   'pass' => 'se_ret'
);
$files = array(
   array(
       'name' => 'uimg',
```

```
        'type' => 'image/jpeg',
        'file' => './profile.jpg',
    )
);

$response = http_post_fields("http://www.example.com/", $fields, $files);
?>
```

http_put_data

http_put_data -- Perform PUT request with data

Description

string **http_put_data** (string `$url` [, string `$data` [, array `$options` [, array `&$info`]]])

Performs an HTTP PUT request on the supplied url.

See the full list of request options.

Parameters

`url`
: URL

`data`
: PUT request body

`options`
: request options

`info`
: Request/response information

Return Values

Returns the HTTP response(s) as string on success, or **FALSE** on failure.

http_put_file

http_put_file -- Perform PUT request with file

Description

string **http_put_file** (string `$url` [, string `$file` [, array `$options` [, array `&$info`]]])

Performs an HTTP PUT request on the supplied url.

See the full list of request options.

Parameters

`url`
: URL

`file`
: The file to put

`options`
: request options

`info`
: Request/response information

Return Values

Returns the HTTP response(s) as string on success, or **FALSE** on failure.

http_put_stream

http_put_stream -- Perform PUT request with stream

Description

string **http_put_stream** (string `$url` [, resource `$stream` [, array `$options` [, array `&$info`]]])

Performs an HTTP PUT request on the supplied url.

See the full list of request options.

Parameters

`url`
: URL

`stream`
: The stream to read the PUT request body from

`options`
: request options

`info`
: Request/response information

Return Values

Returns the HTTP response(s) as string on success, or **FALSE** on failure.

http_request_body_encode

http_request_body_encode -- Encode request body

Description

string **http_request_body_encode** (array `$fields`, array `$files`)

Generate x-www-form-urlencoded resp. form-data encoded request body.

Parameters

`fields`
: POST fields

`files`
: POST files

Return Values

Returns encoded string on success, or FALSE on failure

http_request_method_exists

http_request_method_exists -- Check whether request method exists

Description

int **http_request_method_exists** (mixed `$method`)

Check if a request method is registered (or available by default).

Parameters

`method`
: request method name or ID

Return Values

Returns **TRUE** if the request method is known, else **FALSE**.

http_request_method_name

http_request_method_name -- Get request method name

Description

string **http_request_method_name** (int `$method`)

Get the literal string representation of a standard or registered request method.

Parameters

`method`
: request method ID

Return Values

Returns the request method name as string on success, or **FALSE** on failure.

http_request_method_register

http_request_method_register -- Register request method

Description

int **http_request_method_register** (string `$method`)

Register a custom request method.

Parameters

`method`
: the request method name to register

Return Values

Returns the ID of the request method on success, or FALSE on failure.

http_request_method_unregister

http_request_method_unregister -- Unregister request method

Description

bool **http_request_method_unregister** (mixed `$method`)

Unregister a previously registered custom request method.

Parameters

`method`
: The request method name or ID

Return Values

Returns **TRUE** on success or **FALSE** on failure.

http_request

http_request -- Perform custom request

Description

string **http_request** (int `$method` [, string `$url` [, string `$body` [, array `$options` [, array `&$info`]]]])

Performs a custom HTTP request on the supplied url.

See the full list of request options.

Parameters

`method`
: Request method

`url`
: URL

`body`
: Request body

`options`
: request options

`info`
: Request/response information

Return Values

Returns the HTTP response(s) as string on success, or **FALSE** on failure.

http_redirect

http_redirect -- Issue HTTP redirect

Description

void **http_redirect** ([string `$url` [, array `$params` [, bool `$session = FALSE` [, int `$status`]]]])

Redirect to the given url.

The supplied *`url`* will be expanded with http_build_url(), the *`params`* array will be treated with http_build_str() and the session identification will be appended if *`session`* is true. The HTTP response code will be set according to *`status`*. You can use one of the redirect constants for convenience. Please see » RFC 2616 for which redirect response code to use in which situation. By default PHP will decide which response status fits best.

To be RFC compliant, "Redirecting to <a>URL." will be displayed, if the client doesn't redirect immediately, and the request method was another one than HEAD.

A log entry will be written to the redirect log, if the INI setting http.log.redirect is set and the redirect attempt was successful.

Parameters

`url`
: the URL to redirect to

`params`
: associative array of query parameters

`session`
: whether to append session information

`status`
: custom response status code

Return Values

Returns **FALSE** or *exits* on success with the specified redirection status code. See the INI setting http.force_exit for what "exits" means.

Examples

Example #25 - A http_redirect() example

```
<?php
http_redirect("relpath", array("name" => "value"), true,
HTTP_REDIRECT_PERM);
?>
```

The above example will output:

```
HTTP/1.1 301 Moved Permanently
X-Powered-By: PHP/5.2.2
Content-Type: text/html
Location: http://www.example.com/curdir/relpath?name=value&PHPSESSID=abc

Redirecting to <a
href="http://www.example.com/curdir/relpath?name=value&PHPSESSID=abc">http:/
/www.example.com/curdir/relpath?name=value&PHPSESSID=abc</a>.
```

See Also

- the HttpResponse class if you are using PHP 5.1.0 and above

http_send_content_disposition

http_send_content_disposition -- Send Content-Disposition

Description

bool **http_send_content_disposition** (string `$filename` [, bool `$inline = FALSE`])

Send the *Content-Disposition*. The *Content-Disposition* header is very useful if the data actually being sent came from a file or something similar, that should be "saved" by the client/user (i.e. by the browser's "Save as..." popup window).

Note
This function should be used in conjunction with http_send_data(), http_send_file() and http_send_stream().

Parameters

`filename`
: the file name the "Save as..." dialog should display

`inline`
: if set to **TRUE** and the user agent knows how to handle the content type, it will probably not cause the popup window to be shown

Return Values

Returns **TRUE** on success or **FALSE** on failure.

See Also

- http_send_data()
- http_send_file()
- http_send_stream()
- the HttpResponse class if you are using PHP 5.1.0 and above

http_send_content_type

http_send_content_type -- Send Content-Type

Description

bool **http_send_content_type** ([string `$content_type = 'application/x-octetstream'`])

Send the *Content-Type* of the sent entity.

Note
This function should be used in conjunction with http_send_data(), http_send_file() and http_send_stream().

Parameters

`content_type`
: the desired content type (primary/secondary)

Return Values

Returns **TRUE** on success or **FALSE** on failure.

Errors/Exceptions

Raises an **E_WARNING** when the *`content_type`* doesn't seem to contain a primary and a secondary part.

See Also

- http_send_data()
- http_send_file()
- http_send_stream()
- the HttpResponse class if you are using PHP 5.1.0 and above

http_send_data

http_send_data -- Send arbitrary data

Description

bool **http_send_data** (string `$data`)

Sends raw data with support for (multiple) range requests.

Parameters

`data`
: data to send

Return Values

Returns **TRUE** on success or **FALSE** on failure.

See Also

- http_send_file()
- http_send_stream()
- http_throttle()
- http_send_content_type()
- http_send_content_disposition()
- the HttpResponse class if you are using PHP 5.1.0 and above

http_send_file

http_send_file -- Send file

Description

bool **http_send_file** (string `$file`)

Sends a file with support for (multiple) range requests.

This functions behaviour and further action is dependent on the following INI setting s: http.send.not_found_404 and http.log.not_found.

If the INI setting http.send.not_found_404 is enabled and the INI setting http.log.not_found points to a writable file, a log message is written when the *`file`* was not found.

Parameters

`file`
the file to send

Return Values

Returns **TRUE** on success or **FALSE** on failure.

Examples

Example #26 - A http_send_file() example

```
<?php
http_send_content_disposition("document.pdf", true);
http_send_content_type("application/pdf");
http_throttle(0.1, 2048);
http_send_file("../report.pdf");
?>
```

The above example will output:

```
HTTP/1.1 206 Partial Content
X-Powered-By: PHP/5.2.2
Accept-Ranges: bytes
Content-Length: 12345
Content-Range: bytes 0-12344
Content-Type: application/pdf
Content-Disposition: inline; filename="document.pdf"

%PDF...
```

See Also

- http_send_data()
- http_send_stream()
- http_throttle()
- http_send_content_type()
- http_send_content_disposition()
- the HttpResponse class if you are using PHP 5.1.0 and above

http_send_last_modified

http_send_last_modified -- Send Last-Modified

Description

bool **http_send_last_modified** ([int `$timestamp`])

Send a *Last-Modified* header with a valid HTTP date.

Note
This function should be used in conjunction with http_send_data(), http_send_file() and http_send_stream().

Parameters

`timestamp`
: a Unix timestamp, converted to a valid HTTP date; if omitted, the current time will be sent

Return Values

Returns **TRUE** on success or **FALSE** on failure.

See Also

- the HttpResponse class if you are using PHP 5.1.0 and above

http_send_status

http_send_status -- Send HTTP response status

Description

bool **http_send_status** (int `$status`)

Send HTTP status code.

Parameters

`status`
: HTTP status code (100-599)

Return Values

Returns **TRUE** on success or **FALSE** on failure.

See Also

- the HttpResponse class if you are using PHP 5.1.0 and above

http_send_stream

http_send_stream -- Send stream

Description

bool **http_send_stream** (resource `$stream`)

Sends an already opened stream with support for (multiple) range requests.

Parameters

`stream`
: stream to read from (must be seekable)

Return Values

Returns **TRUE** on success or **FALSE** on failure.

See Also

- http_send_data()
- http_send_file()
- http_throttle()
- http_send_content_type()
- http_send_content_disposition()
- the HttpResponse class if you are using PHP 5.1.0 and above

http_throttle

http_throttle -- HTTP throttling

Description

void **http_throttle** ([float `$sec` [, int `$bytes = 40960`]])

Sets the throttle delay and send buffer size.

Note
This function should be used in conjunction with http_send_data(), http_send_file() and http_send_stream().

Note
Provides a basic throttling mechanism, which will yield the current process or thread until the entity has been completely sent.

Note
This may not work as expected with the following SAPI(s): FastCGI.

Parameters

`sec`
: seconds to sleep after each chunk sent

`bytes`
: the chunk size in bytes

Examples

Example #27 - A http_throttle() example

Send file with approximately 20 kbyte/s.

```
<?php
```

```
// ~ 20 kbyte/s
# http_throttle(1, 20000);
# http_throttle(0.5, 10000);
http_throttle(0.1, 2000);
http_send_file('document.pdf');
?>
?>
```

See Also

- http_send_data()
- http_send_file()
- http_send_stream()
- the HttpResponse class if you are using PHP 5.1.0 and above

http_build_str

http_build_str -- Build query string

Description

string **http_build_str** (array `$query` [, string `$prefix` [, string `$arg_separator`]])

Opponent to parse_str().

Parameters

`query`
: associative array of query string parameters

`prefix`
: top level prefix

`arg_separator`
: argument separator to use (by default the INI setting arg_separator.output will be used, or "&" if neither is set

Return Values

Returns the built query as string on success or **FALSE** on failure.

See Also

- standard http_build_query()
- http_build_url()

http_build_url

http_build_url -- Build an URL

Description

string **http_build_url** ([mixed `$url` [, mixed `$parts` [, int `$flags = HTTP_URL_REPLACE` [, array `&$new_url`]]]])

Build an URL.

The parts of the second URL will be merged into the first according to the flags argument.

Parameters

`url`
(part(s) of) an URL in form of a string or associative array like parse_url() returns

`parts`
same as the first argument

`flags`
a bitmask of binary or'ed HTTP_URL constants; **HTTP_URL_REPLACE** is the default

`new_url`
if set, it will be filled with the parts of the composed url like parse_url() would return

Return Values

Returns the new URL as string on success or **FALSE** on failure.

Examples

Example #28 - A http_build_url() example

```
<?php
echo http_build_url("http://user@www.example.com/pub/index.php?a=b#files",
   array(
       "scheme" => "ftp",
       "host" => "ftp.example.com",
       "path" => "files/current/",
       "query" => "a=c"
   ),
   HTTP_URL_STRIP_AUTH | HTTP_URL_JOIN_PATH | HTTP_URL_JOIN_QUERY |
HTTP_URL_STRIP_FRAGMENT
);
```

```
?>
```

The above example will output:

```
ftp://ftp.example.com/pub/files/current/?a=b&a=c
```

See Also

- parse_url()
- http_build_str()