Instantiation

mime

acceptable

send

receive

progress

_curl

_stream

_socket

engine

subst

request

minify

rss

whois

slug

diacritics

filler

Web

The Web class (and its descendants, Geo (geo) and Pingback (pingback)) contains several helpers to interact with HTTP clients and servers.

Namespace: `\`
File location: `lib/web.php`

Instantiation

Return class instance

```
$web = \Web::instance();
```

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

mime

Detect MIME type using file extension

```
string mime ( string $file )
```

Detects MIME type by comparing the file extension against a predefined array. Use this as a lightweight alternative to Fileinfo, which is not available on some server setups and would be more costly in terms of performance. In the following example, we see how to set the right response header content type to display an image by PHP.

```
$web = \Web::instance();
$file = 'ui/images/south-park.jpg';
$mime = $web->mime($file); // returns 'image/jpeg'

header('Content-Type: '.$mime);
echo $f3->read($file);
```

If an extension is not known to this method, it will return 'application/octet-stream'.

acceptable

Returns the list of acceptable MIME types of the browser.

```
array|string|false acceptable ( [ string|array $list = NULL ] )
```

It returns the MIME types stated by the browser in the HTTP Accept header as an array.

```
print_r(Web::instance()->acceptable());

/*
Returns for example:

Array
(
    [text/html] => 1
    [application/xhtml+xml] => 1
    [application/xml] => 0.9
)
*/
```

If a list of MIME types is specified, it returns the best match, or FALSE if none found.

```
$web->acceptable(array('application/xml','application/xhtml+xml')); // returns applic
ation/xhtml+xml
```

send

Transmits a file to the client and returns the file size on success

```
int|false send ( string $file [, string $mime = NULL [, int $kbps = 0 [, bool $force
= TRUE [, $name = NULL [, $flush = TRUE ]]]]] )
```

The `$file` is a filename that must validate against the PHP function `is_file($file)`.

With the `$mime` argument you can explicitly set a mime type. Leave it to `NULL` to let the framework detect the MIME type using the `$file` file extension.

The 3rd argument `$kbps` specify the throttle speed, measured in Kilobits per second. It allows you to actively limit users' download rates, to avoid overloading your server when you send big multimedia files for example.

Use `$force` to add a `Content-Disposition: attachment;` HTTP header that will force the browser to download the file instead of trying to display it (as it is often the case with well-supported filetypes such as .txt, .jpg, .png, .git, .css, .js, .pdf, .mp3, etc.).

When `$force` is set to `TRUE`, use `$name` to specify the name of the downloaded file (otherwise `$file` will be used).

When `$flush` is set to `TRUE`, the output is flushed every 1 KiB. In most cases, you shouldn't need to disable this. However, it may come in handy when running tests.

Usage example:

```
$throttle = 2048; // throttle to around 256 KB /s
$sent = $web->send('data/documents.zip', NULL, $throttle);
if ( !$sent)  { /*error*/ }
```

Send/Receive example:

```
$file=$f3->get('UI').'images/wallpaper.jpg';
ob_start();
// send the file without any download dialog
$web->send($file,NULL,512,FALSE);
$out=ob_get_clean();

// setup a route and its associated handler
$f3->set('UPLOADS',$f3->get('TEMP'));
$f3->route('PUT /upload/@filename',
    function() use($web) { $web->receive(); }
);
// mock a request that will actually upload the file
$f3->mock('PUT /upload/'.basename($file),NULL,NULL,$f3->read($file));

if (is_file($target=$f3->get('UPLOADS').basename($file)))
    echo 'Uploaded file done via PUT';

@unlink($target);
```

receive

Receive and process files from a client sent via PUT or POST.

```
array|bool receive ( [ callback $func = NULL [, bool $overwrite = FALSE [, callback|b
ool $slug = TRUE ]]] )
```

This function has two behaviours:

1. on a POST request, it fetches files uploaded via HTML file inputs and move them into the directory specified in UPLOADS (quick-reference#uploads) system var. The returned value is an array containing the upload status for each uploaded file.
2. on a PUT request, it writes the contents of the request body into a target file located in the directory specified in UPLOADS (quick-reference#uploads) system var.

You can set an optional upload handler function to validate uploaded files, before moving them to their desired location. Lets have a look at this:

```
$f3->set('UPLOADS','uploads/'); // don't forget to set an Upload directory, and make
it writable!

$overwrite = false; // set to true, to overwrite an existing file; Default: false
$slug = true; // rename file to filesystem-friendly version

$files = $web->receive(function($file,$formFieldName){
        var_dump($file);
        /* looks like:
          array(5) {
              ["name"] =>     string(19) "csshat_quittung.png"
              ["type"] =>     string(9) "image/png"
              ["tmp_name"] => string(14) "/tmp/php2YS85Q"
              ["error"] =>    int(0)
              ["size"] =>     int(172245)
            }
        */
        // $file['name'] already contains the slugged name now

        // maybe you want to check the file size
        if($file['size'] > (2 * 1024 * 1024)) // if bigger than 2 MB
            return false; // this file is not valid, return false will skip moving it

        // everything went fine, hurray!
        return true; // allows the file to be moved from php tmp dir to your defined
upload dir
    },
    $overwrite,
    $slug
);

var_dump($files);
/* looks like:
  array(3) {
      ["uploads/csshat_quittung.png"] => bool(true)
      ["uploads/foo.pdf"] => bool(false)
      ["uploads/my.pdf"] => bool(true)
    }
  foo.pdf was not uploaded...
*/
```

Notice: Having trouble to get this working? Don't forget to set the **enctype="multipart/form-data"** attribute in your **<form>** tag.

A callback can also be another function or class method. Have a look at the call() (base#call) function description to see all possibilities.

```
function callback() {
    echo 'file uploaded!';
}

Web::instance()->receive('callback', false, true);
```

Notice: The callback function will be called for every successful upload if specified.

You can also generate a custom file name for the uploaded files using a callback function in the `slug` parameter:

```
$files = $web->receive(function($file,$formFieldName){
        // ...
    },true,function($fileBaseName, $formFieldName){
        // build new file name from base name or input field name
        return "custom_filename.jpg";
    }
);
```

progress

Returns the progress of a file upload if `session.upload_progress.enabled` is set to `1` in the php.ini

```
int|false Web::instance()->progress ( string $session_id )
```

The `$session_id` is the "key" of the `$_SESSION` array and is necessary to retrieve the status of a specific user. Please read the PHP Docs (http://php.net/manual/session.upload-progress.php) for more information.

_curl

Perform HTTP requests via cURL. It's a protected method and is used by `Web->request()`.

Returns an array containing the content and the header.

_stream

Perform HTTP requests via PHP stream wraper. It's a protected method and is used by `Web->request()`.

Returns an array containing the content and the header.

_socket

Perform a HTTP request on low-level via TCP/IP sockets. It's a protected method and is used by `Web->request()`.

Returns an array containing the content and the header.

engine

Specify the HTTP request engine to use

```
string engine ( [ string $arg = 'curl' ] )
```

Sets the engine to be used by `Web->request()`. If the selected engine is not available, it falls back to an applicable substitute. Possible values are:

- curl (default)
- stream
- socket

Notice: The cURL and stream wrapper engines need their appropriate php extension to be installed and activated to work properly. Otherwise, sockets are used.

subst

Replace old headers with new elements

```
NULL subst ( array &$old, string|array $new )
```

request

Perform a HTTP request

```
array|false request ( string $url [, array $options = NULL ] )
```

Sends a HTTP request to a server and returns an array containing the content and the header. The requested page will be cached as instructed by the remote server. Here's an example of a simple GET request to download a remote file:

```
var_dump( \Web::instance()->request('http://www.golem.de/1303/98339-55766-i_rc.jpg'
));
/* returns:

array(4) {
    'body' => string " IMAGE BINARY DATA " (length=7761)
    'headers' =>
        array (size=14)
        0 => string 'HTTP/1.1 200 OK' (length=15)
        1 => string 'Server: nginx' (length=13)
        2 => string 'Date: Thu, 18 Dec 2013 12:40:11 GMT' (length=35)
        3 => string 'Content-Type: image/jpeg' (length=24)
        4 => string 'Content-Length: 7761' (length=20)
        5 => string 'Connection: close' (length=17)
        6 => string 'Keep-Alive: timeout=3' (length=21)
        7 => string 'Last-Modified: Fri, 22 Mar 2013 09:41:02 GMT' (length=44)
        8 => string 'ETag: "514c272e-1e51"' (length=21)
        9 => string 'X-UPSTREAM: www1.golem.de' (length=25)
        10 => string 'Expires: Sun, 18 Jan 2014 12:40:11 GMT' (length=38)
        11 => string 'Cache-Control: max-age=2678400' (length=30)
        12 => string 'X-Cache-Status: HIT' (length=19)
        13 => string 'Accept-Ranges: bytes' (length=20)
    'engine' => string 'cURL' (length=4)
    'cached' => boolean false
    'error' => string '' (length=0)
}
*/
```

$options

You can use HTTP context options (http://www.php.net/manual/en/context.http.php) to configure the request for your needs. This way you can build request of every type, including the usage of cookies or auth mechanisms, proxy, user-agent, and so on. Here are some samples:

```
$options = array(
    'timeout' => 6, // change request timeout
    'header' => [
        'Cookie: User=1;Foo='.rawurlencode('bar'), // add cookie
        'Authorization: Basic '.base64_encode('user:password'), // auth header
    ],
    'proxy'=> 'http://186.227.8.21:3128', // use proxy, protocols: http, https, socks
4, socks4a, socks5, socks5h
);
```

GET

Perform a GET request with URL parameters:

```
$url = 'http://www.mydomain.com/index.php';
$params = array(
    'parameter1' => 'value1',
    'parameter2' => 'value2'
);
$options = array('method' => 'GET');
$url .= '?'.http_build_query($params);

$result = \Web::instance()->request($url, $options);
```

POST

Perform a POST request with POST data.

```
$url = 'http://www.mydomain.com/index.php';

$postVars = array(
    'parameter1' => 'value1',
    'parameter2' => 'value2'
);
$options = array(
    'method'  => 'POST',
    'content' => http_build_query($postVars),
);
$result = \Web::instance()->request($url, $options);
```

PUT

Upload a file via PUT request.

```
$f3 = \Base::instance();
$web = \Web::instance();

$url = 'http://www.mydomain.com/upload/';
$file = '/path/to/myFile.zip';

$options = array(
    'method'  => 'PUT',
    'content' => $f3->read($file),
    'header' => array('Content-Type: '.$web->mime($file));
);

$result = $web->request($url, $options);
```

minify

Minify CSS and Javascript files by stripping whitespaces and comments. Returns a combined output as a string.

```
string minify ( string|array $files [, string $mime = NULL [, bool $header = TRUE [,
string $path = NULL ]]] )
```

Example:

```
$minified = Web::instance()->minify('style.css,framework.css,null.css');
```

This method will also auto-detect the mime-type of the files that are going to be minified and send as header Content-Type to the browser. You can overwrite the used mime-type with the `$mime` parameter, and stop sending the header data entirely by setting the `$header` parameter to `FALSE`.

If the files are not located within one of the UI (quick-reference#ui) search paths, you must use the `$path` argument to specify their directory path. In particular, set `$path=''` if the provided file paths are absolute.

To get maximum performance, you can enable the F3 system caching (quick-reference#cache) and F3 will use it to save/retrieve file(s) to minify and to save the combined output as well. You can have a look at the Cache Engine User Guide (optimization#cache-engine) for more details. . To see an example of how `minify` can be used in your templates, check the Keeping Javascript and CSS on a Healthy Diet (optimization#keeping-javascript-and-css-on-a-healthy-diet) section of the User Guide.

rss

Parse RSS feed and optionally return an array of all tags.

```
array|false rss ( string $url [, int $max = 10 [, string $tags = NULL ]] )
```

Example:

```
$count = 20; // Default: 10
$tags = 'title,link,pubDate'; // Default: NULL (all tags)

Web::instance()->rss('http://example.org/feed.rss', $count, $tags);
```

whois

Retrieve information from whois server

```
string|false whois ( string $addr [, string $server = 'whois.internic.net'] )
```

example:

```
echo $web->whois('fatfreeframework.com');

/* returns:
Whois Server Version 2.0

Domain names in the .com and .net domains can now be registered
with many different competing registrars. Go to http://www.internic.net
for detailed information.

 .  Domain Name: FATFREEFRAMEWORK.COM
   Registrar: INTERNETWORX LTD. & CO. KG
   Whois Server: whois.domrobot.com
   Referral URL: http://www.domrobot.com
   Name Server: NS.INWX.DE
   Name Server: NS2.INWX.DE
   Name Server: NS3.INWX.EU
...
*/
```

slug

Method to return a URL- and filesystem-friendly string.

```
string slug ( string $text )
```

The purpose of this function is to convert foreign characters to their approximate English keyboard character equivalents. Therefor it uses ISO-9 transliteration with a lookup table array (web#diacritics) that can be extended with the DIACRITICS (quick-reference#diacritics) var. Furthermore, it is designed to remove all non-alphanumeric characters and convert them to dashes.

```
echo Web::instance()->slug('ĤÈĹĹŌ'); // displays 'hello'
echo Web::instance()->slug('Ein schöner Artikel über Max & John!'); // displays 'ein-
schoener-artikel-ueber-max-john'
```

diacritics

Return the preset diacritics translation table

```
array diacritics ( )
```

This table is used by the slug() method (web#slug) in conjunction with the DIACRITICS (quick-reference#diacritics) variable.

filler

Return a chunk of text from the standard Lorem Ipsum passage

```
string filler ( [ int $count = 1 [, int $max = 20 [, bool $std = TRUE ]]] )
```

This function might be useful to fill your empty layout with some placeholder text. Therefore `$count` controls the number of sentences being created with a randomly amount of words between 3 and `$max`. By default, the `$std` var specifies if the returned text will start with a default `Lorem ipsum dolor` … filler made of 124 chars of length.

```
echo $web->filler(3, 5, FALSE);
/* returns 3 random sentences, with a maximum of 5 words, similar to this:
'Iste corporis aut. Exercitationem corporis rem harum repellat. Cupiditate eligendi d
ebitis.'
*/
```