URLs

Introduction

Dealing with URL strings: encoding, decoding and parsing.

Installing/Configuring

Requirements

No external libraries are needed to build this extension.

Installation

There is no installation needed to use these functions; they are part of the PHP core.

Runtime Configuration

This extension has no configuration directives defined in *php.ini*.

Resource Types

This extension has no resource types defined.

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.

The following constants are meant to be used with parse_url() and are available since PHP 5.1.2.

PHP_URL_SCHEME (integer)

PHP_URL_HOST (integer)

PHP_URL_PORT (integer)

PHP_URL_USER (integer)

PHP_URL_PASS (integer)

PHP_URL_PATH (integer)

PHP_URL_QUERY (integer)

PHP_URL_FRAGMENT (integer)

URL Functions

base64_decode

base64_decode -- Decodes data encoded with MIME base64

Description

string **base64_decode** (string `$data` [, bool `$strict`])

Decodes a base64 encoded *`data`*.

Parameters

`data`
: The decoded data.

`strict`
: Returns **FALSE** if input contains space or some other separator.

Return Values

Returns the original data or **FALSE** on failure. The returned data may be binary.

ChangeLog

Version	Description
5.2.0	*`strict`* added

Examples

Example #1 - base64_decode() example

```
<?php
$str = 'VGhpcyBpcyBhbiBlbmNvZGVkIHN0cmluZw==';
echo base64_decode($str);
?>
```

The above example will output:

```
This is an encoded string
```

See Also

- base64_encode()
- » RFC 2045 section 6.8

base64_encode

base64_encode -- Encodes data with MIME base64

Description

string **base64_encode** (string `$data`)

Encodes the given *`data`* with base64.

This encoding is designed to make binary data survive transport through transport layers that are not 8-bit clean, such as mail bodies.

Base64-encoded data takes about 33% more space than the original data.

Parameters

`data`
The data to encode.

Return Values

The encoded data, as a string.

Examples

Example #2 - base64_encode() example

```
<?php
$str = 'This is an encoded string';
echo base64_encode($str);
?>
```

The above example will output:

```
VGhpcyBpcyBhbiBlbmNvZGVkIHN0cmluZw==
```

See Also

- base64_decode()
- chunk_split()

- convert_uuencode()
- » RFC 2045 section 6.8

get_headers

get_headers -- Fetches all the headers sent by the server in response to a HTTP request

Description

array **get_headers** (string `$url` [, int `$format`])

get_headers() returns an array with the headers sent by the server in response to a HTTP request.

Parameters

`url`
The target URL.

`format`
If the optional *`format`* parameter is set to 1, get_headers() parses the response and sets the array's keys.

Return Values

Returns an indexed or associative array with the headers, or **FALSE** on failure.

ChangeLog

Version	Description
5.1.3	This function now uses the default stream context, which can be set/changed with the stream_context_get_default() function.

Examples

Example #3 - get_headers() example

```
<?php
$url = 'http://www.example.com';

print_r(get_headers($url));
```

```
print_r(get_headers($url, 1));
?>
```

The above example will output something similar to:

```
Array
(
   [0] => HTTP/1.1 200 OK
   [1] => Date: Sat, 29 May 2004 12:28:13 GMT
   [2] => Server: Apache/1.3.27 (Unix)  (Red-Hat/Linux)
   [3] => Last-Modified: Wed, 08 Jan 2003 23:11:55 GMT
   [4] => ETag: "3f80f-1b6-3e1cb03b"
   [5] => Accept-Ranges: bytes
   [6] => Content-Length: 438
   [7] => Connection: close
   [8] => Content-Type: text/html
)

Array
(
   [0] => HTTP/1.1 200 OK
   [Date] => Sat, 29 May 2004 12:28:14 GMT
   [Server] => Apache/1.3.27 (Unix)  (Red-Hat/Linux)
   [Last-Modified] => Wed, 08 Jan 2003 23:11:55 GMT
   [ETag] => "3f80f-1b6-3e1cb03b"
   [Accept-Ranges] => bytes
   [Content-Length] => 438
   [Connection] => close
   [Content-Type] => text/html
)
```

get_meta_tags

get_meta_tags -- Extracts all meta tag content attributes from a file and returns an array

Description

array **get_meta_tags** (string `$filename` [, bool `$use_include_path`])

Opens *`filename`* and parses it line by line for <meta> tags in the file. The parsing stops at *</head>*.

Parameters

`filename`
The path to the HTML file, as a string. This can be a local file or an URL.

Example #4 - What get_meta_tags() parses

```
<meta name="author" content="name">
<meta name="keywords" content="php documentation">
<meta name="DESCRIPTION" content="a php manual">
<meta name="geo.position" content="49.33;-86.59">
</head> <!-- parsing stops here -->
```

(pay attention to line endings - PHP uses a native function to parse the input, so a Mac file won't work on Unix).

`use_include_path`
Setting *`use_include_path`* to **TRUE** will result in PHP trying to open the file along the standard include path as per the include_path directive. This is used for local files, not URLs.

Return Values

Returns an array with all the parsed meta tags.

The value of the name property becomes the key, the value of the content property becomes the value of the returned array, so you can easily use standard array functions to traverse it or access single values. Special characters in the value of the name property are substituted with '_', the rest is converted to lower case. If two meta tags have the same name, only the last one is returned.

ChangeLog

Version	Description

4.0.5	Support for unquoted HTML attributes was added.

Examples

Example #5 - What get_meta_tags() returns

```
<?php
// Assuming the above tags are at www.example.com
$tags = get_meta_tags('http://www.example.com/');

// Notice how the keys are all lowercase now, and
// how . was replaced by _ in the key.
echo $tags['author'];       // name
echo $tags['keywords'];     // php documentation
echo $tags['description'];  // a php manual
echo $tags['geo_position']; // 49.33;-86.59
?>
```

See Also

- htmlentities()
- urlencode()

http_build_query

http_build_query -- Generate URL-encoded query string

Description

string **http_build_query** (array `$formdata` [, string `$numeric_prefix` [, string `$arg_separator`]])

Generates a URL-encoded query string from the associative (or indexed) array provided.

Parameters

`formdata`
: May be an array or object containing properties. The array form may be a simple one-dimensional structure, or an array of arrays (who in turn may contain other arrays).

`numeric_prefix`
: If numeric indices are used in the base array and this parameter is provided, it will be prepended to the numeric index for elements in the base array only. This is meant to allow for legal variable names when the data is decoded by PHP or another CGI application later on.

`arg_separator`
: arg_separator.output is used to separate arguments, unless this parameter is specified, and is then used.

Return Values

Returns a URL-encoded string.

ChangeLog

Version	Description
5.1.2	The *`arg_separator`* parameter was added.
5.1.3	Square brackets are escaped.

Examples

Example #6 - Simple usage of http_build_query()

```
<?php
$data = array('foo'=>'bar',
              'baz'=>'boom',
              'cow'=>'milk',
              'php'=>'hypertext processor');

echo http_build_query($data); //
foo=bar&baz=boom&cow=milk&php=hypertext+processor
echo http_build_query($data, '', '&amp;'); //
foo=bar&amp;baz=boom&amp;cow=milk&amp;php=hypertext+processor

?>
```

Example #7 - http_build_query() with numerically index elements.

```
<?php
$data = array('foo', 'bar', 'baz', 'boom', 'cow' => 'milk', 'php'
=>'hypertext processor');

echo http_build_query($data) . "\n";
echo http_build_query($data, 'myvar_');
?>
```

The above example will output:

```
0=foo&1=bar&2=baz&3=boom&cow=milk&php=hypertext+processor
myvar_0=foo&myvar_1=bar&myvar_2=baz&myvar_3=boom&cow=milk&php=hypertext+proc
essor
```

Example #8 - http_build_query() with complex arrays

```
<?php
$data = array('user'=>array('name'=>'Bob Smith',
                            'age'=>47,
                            'sex'=>'M',
                            'dob'=>'5/12/1956'),
              'pastimes'=>array('golf', 'opera', 'poker', 'rap'),
              'children'=>array('bobby'=>array('age'=>12,
                                               'sex'=>'M'),
                                'sally'=>array('age'=>8,
                                               'sex'=>'F')),
              'CEO');

echo http_build_query($data, 'flags_');
?>
```

this will output : (word wrapped for readability)

```
user%5Bname%5D=Bob+Smith&user%5Bage%5D=47&user%5Bsex%5D=M&
user%5Bdob%5D=5%2F12%2F1956&pastimes%5B0%5D=golf&pastimes%5B1%5D=opera&
pastimes%5B2%5D=poker&pastimes%5B3%5D=rap&children%5Bbobby%5D%5Bage%5D=12&
```

```
children%5Bbobby%5D%5Bsex%5D=M&children%5Bsally%5D%5Bage%5D=8&
children%5Bsally%5D%5Bsex%5D=F&flags_0=CEO
```

Note

Only the numerically indexed element in the base array "CEO" received a prefix. The other numeric indices, found under pastimes, do not require a string prefix to be legal variable names.

Example #9 - Using http_build_query() with an object

```
<?php
class myClass {
   var $foo;
   var $baz;

   function myClass() {
       $this->foo = 'bar';
       $this->baz = 'boom';
   }
}

$data = new myClass();

echo http_build_query($data); // foo=bar&baz=boom

?>
```

See Also

- parse_str()
- parse_url()
- urlencode()
- array_walk()

parse_url

parse_url -- Parse a URL and return its components

Description

mixed **parse_url** (string `$url` [, int `$component`])

This function parses a URL and returns an associative array containing any of the various components of the URL that are present.

This function is *not* meant to validate the given URL, it only breaks it up into the above listed parts. Partial URLs are also accepted, parse_url() tries its best to parse them correctly.

Parameters

`url`
: The URL to parse

`component`
: Specify one of **PHP_URL_SCHEME**, **PHP_URL_HOST**, **PHP_URL_PORT**, **PHP_URL_USER**, **PHP_URL_PASS**, **PHP_URL_PATH**, **PHP_URL_QUERY** or **PHP_URL_FRAGMENT** to retrieve just a specific URL component as a string.

Return Values

On seriously malformed URLs, parse_url() may return **FALSE** and emit a **E_WARNING**. Otherwise an associative array is returned, whose components may be (at least one):

- *scheme* - e.g. http
- *host*
- *port*
- *user*
- *pass*
- *path*
- *query* - after the question mark *?*
- *fragment* - after the hashmark #

If the *`component`* parameter is specified a string is returned instead of an array.

ChangeLog

Version	Description
5.1.2	Added the *component* parameter

Examples

Example #10 - A parse_url() example

```
<?php
$url = 'http://username:password@hostname/path?arg=value#anchor';

print_r(parse_url($url));

echo parse_url($url, PHP_URL_PATH);
?>
```

The above example will output:

```
Array
(
    [scheme] => http
    [host] => hostname
    [user] => username
    [pass] => password
    [path] => /path
    [query] => arg=value
    [fragment] => anchor
)
/path
```

Notes

Note

This function doesn't work with relative URLs.

Note

This function is intended specifically for the purpose of parsing URLs and not URIs. However, to comply with PHP's backwards compatibility requirements it makes an exception for the file:// scheme where tripple slashes (file:///...) are allowed. For any other scheme this is invalid.

See Also

- pathinfo()
- parse_str()
- http_build_query()
- dirname()
- basename()

rawurldecode

rawurldecode -- Decode URL-encoded strings

Description

string **rawurldecode** (string `$str`)

Returns a string in which the sequences with percent (%) signs followed by two hex digits have been replaced with literal characters.

Parameters

`str`
The URL to be decoded.

Return Values

Returns the decoded URL, as a string.

Examples

Example #11 - rawurldecode() example

```
<?php

echo rawurldecode('foo%20bar%40baz'); // foo bar@baz

?>
```

Notes

Note

rawurldecode() does not decode plus symbols ('+') into spaces. urldecode() does.

See Also

- rawurlencode()

- urldecode()
- urlencode()

rawurlencode

rawurlencode -- URL-encode according to RFC 1738

Description

string **rawurlencode** (string `$str`)

Encodes the given string according to » RFC 1738.

Parameters

`str`
The URL to be encoded.

Return Values

Returns a string in which all non-alphanumeric characters except -_. have been replaced with a percent (%) sign followed by two hex digits. This is the encoding described in » RFC 1738 for protecting literal characters from being interpreted as special URL delimiters, and for protecting URLs from being mangled by transmission media with character conversions (like some email systems).

Examples

Example #12 - including a password in an FTP URL

```
<?php
echo '<a href="ftp://user:', rawurlencode('foo @+%/'),
    '@ftp.example.com/x.txt">';
?>
```

The above example will output:

```
<a href="ftp://user:foo%20%40%2B%25%2F@ftp.example.com/x.txt">
```

Or, if you pass information in a PATH_INFO component of the URL:

Example #13 - rawurlencode() example 2

```
<?php
echo '<a href="http://example.com/department_list_script/',
```

```
   rawurlencode('sales and marketing/Miami'), '">';
?>
```

The above example will output:

```
<a
href="http://example.com/department_list_script/sales%20and%20marketing%2FMi
ami">
```

See Also

- rawurldecode()
- urldecode()
- urlencode()
- » RFC 1738

urldecode

urldecode -- Decodes URL-encoded string

Description

string **urldecode** (string `$str`)

Decodes any % ## encoding in the given string.

Parameters

`str`
The string to be decoded.

Return Values

Returns the decoded string.

Examples

Example #14 - urldecode() example

```
<?php
$a = explode('&', $QUERY_STRING);
$i = 0;
while ($i < count($a)) {
   $b = split('=', $a[$i]);
   echo 'Value for parameter ', htmlspecialchars(urldecode($b[0])),
        ' is ', htmlspecialchars(urldecode($b[1])), "<br />\n";
   $i++;
}
?>
```

See Also

- urlencode()
- rawurlencode()
- rawurldecode()

urlencode

urlencode -- URL-encodes string

Description

string **urlencode** (string `$str`)

This function is convenient when encoding a string to be used in a query part of a URL, as a convenient way to pass variables to the next page.

Parameters

`str`
The string to be encoded.

Return Values

Returns a string in which all non-alphanumeric characters except -_. have been replaced with a percent (%) sign followed by two hex digits and spaces encoded as plus (+) signs. It is encoded the same way that the posted data from a WWW form is encoded, that is the same way as in *application/x-www-form-urlencoded* media type. This differs from the » RFC 1738 encoding (see rawurlencode()) in that for historical reasons, spaces are encoded as plus (+) signs.

Examples

Example #15 - urlencode() example

```
<?php
echo '<a href="mycgi?foo=', urlencode($userinput), '">';
?>
```

Example #16 - urlencode() and htmlentities() example

```
<?php
$query_string = 'foo=' . urlencode($foo) . '&bar=' . urlencode($bar);
echo '<a href="mycgi?' . htmlentities($query_string) . '">';
?>
```

Notes

Note
Be careful about variables that may match HTML entities. Things like &amp, &copy and &pound are parsed by the browser and the actual entity is used instead of the desired variable name. This is an obvious hassle that the W3C has been telling people about for years. The reference is here: » http://www.w3.org/TR/html4/appendix/notes.html#h-B.2.2. PHP supports changing the argument separator to the W3C-suggested semi-colon through the arg_separator .ini directive. Unfortunately most user agents do not send form data in this semi-colon separated format. A more portable way around this is to use & instead of & as the separator. You don't need to change PHP's arg_separator for this. Leave it as &, but simply encode your URLs using htmlentities() or htmlspecialchars().

See Also

- urldecode()
- htmlentities()
- rawurlencode()
- rawurldecode()