

URL Functions

Introduzione

Dealing with URL strings: encoding, decoding and parsing.

Requisiti

Non sono necessarie librerie esterne per utilizzare questo modulo.

Installazione

Non è necessaria nessuna installazione per usare queste funzioni, esse fanno parte del core di PHP.

Configurazione di Runtime

Questa estensione non definisce alcuna direttiva di configurazione in `php.ini`

Tipi di risorse

Questa estensione non definisce alcun tipo di risorsa.

Costanti predefinite

Questa estensione non definisce alcuna costante.

Sommario

[base64_decode](#) -- Decodes data encoded with MIME base64

[base64_encode](#) -- Encodes data with MIME base64

[get_headers](#) -- Fetches all the headers sent by the server in response to a HTTP request

[get_meta_tags](#) -- Extracts all meta tag content attributes from a file and returns an array

[http_build_query](#) -- Generate URL-encoded query string

[parse_url](#) -- Parse a URL and return its components
[rawurldecode](#) -- Decode URL-encoded strings
[rawurlencode](#) -- URL-encode according to RFC 1738
[urldecode](#) -- Decodes URL-encoded string
[urlencode](#) -- URL-encodes string

base64_decode

(PHP 3, PHP 4, PHP 5)

base64_decode -- Decodes data encoded with MIME base64

Description

string **base64_decode** (string encoded_data)

base64_decode() decodes *encoded_data* and returns the original data or **FALSE** on failure. The returned data may be binary.

Esempio 1. base64_decode() example

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

This example will produce:

```
This is an encoded string
```

See also [base64_encode\(\)](#) and [RFC 2045](#) section 6.8.

base64_encode

(PHP 3, PHP 4, PHP 5)

base64_encode -- Encodes data with MIME base64

Description

string **base64_encode** (string data)

base64_encode() returns *data* encoded 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.

Esempio 1. `base64_encode()` example

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

This example will produce:

```
VGhpcyBpcyBhbiBlbmNvZGVkIHNOcm1uZw==
```

See also [base64_decode\(\)](#), [chunk_split\(\)](#), [convert_uuencode\(\)](#) and [RFC 2045](#) section 6.8.

get_headers

(PHP 5)

`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. Returns `FALSE` on failure and an error of level `E_WARNING` will be issued.

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

Esempio 1. `get_headers()` example

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

print_r(get_headers($url));

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

Il precedente esempio visualizzerà qualcosa simile a:

```
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

(PHP 3 >= 3.0.4, PHP 4, PHP 5)

`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. This can be a local file or an URL. The parsing stops at </head>.

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.

Esempio 1. 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).

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.

Esempio 2. 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
?>
```

Nota: As of PHP 4.0.5, `get_meta_tags()` supports unquoted HTML attributes.

See also [htmlentities\(\)](#) and [urlencode\(\)](#).

http_build_query

(PHP 5)

`http_build_query` -- Generate URL-encoded query string

Description

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

Generates a URL-encoded query string from the associative (or indexed) array provided. *formdata* may be an array or object containing properties. A *formdata* array may be a simple one-dimensional structure, or an array of arrays (who in turn may contain other arrays). If numeric indices are used in the base array and a *numeric_prefix* is provided, it will be prepended to the numeric index for elements in the base array only. This is to allow for legal variable names when the data is decoded by PHP or another CGI application later on.

Nota: [arg_separator.output](#) is used to separate arguments.

Esempio 1. 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
```

```
?>
```

Esempio 2. `http_build_query()` with numerically index elements.

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

echo http_build_query($data);
/* Outputs:
    0=foo&1=bar&2=baz&3=boom&cow=milk&php=hypertext+processor
*/

echo http_build_query($data, 'myvar_');
/* Outputs:
    myvar_0=foo&myvar_1=bar&myvar_2=baz&myvar_3=boom&cow=milk&php=hypertext+processor
*/
?>
```

Esempio 3. `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[name]=Bob+Smith&user[age]=47&user[sex]=M&user[dob]=5%1F12%1F1956&
pastimes[0]=golf&pastimes[1]=opera&pastimes[2]=poker&pastimes[3]=rap&
children[bobby][age]=12&children[bobby][sex]=M&children[sally][age]=8&
children[sally][sex]=F&flags_0=CEO
```

Nota: 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.

Esempio 4. 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\(\)](#), and [array_walk\(\)](#)

parse_url

(PHP 3, PHP 4, PHP 5)

parse_url -- Parse a URL and return its components

Descrizione

array **parse_url** (string url)

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.

Elenco dei parametri

url

The URL to parse

Valori restituiti

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 #

Esempi

Esempio 1. A parse_url() example

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

print_r(parse_url($url));
?>
```

Il precedente esempio visualizzerà:

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

Note

Nota: This function doesn't work with relative URLs.

Vedere anche:

[pathinfo\(\)](#)

[parse_str\(\)](#)

[dirname\(\)](#)

[basename\(\)](#)

rawurldecode

(PHP 3, PHP 4, PHP 5)

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.

Esempio 1. rawurldecode() example

```
<?php
echo rawurldecode('foo%20bar%40baz'); // foo bar@baz
?>
```

Nota: **rawurldecode()** does not decode plus symbols ('+') into spaces. [urldecode\(\)](#) does.

See also [rawurlencode\(\)](#), [urldecode\(\)](#) and [urlencode\(\)](#).

rawurlencode

(PHP 3, PHP 4, PHP 5)

rawurlencode -- URL-encode according to RFC 1738

Description

string **rawurlencode** (string str)

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 URL's from being mangled by transmission media with character conversions (like some email systems). For example, if you want to include a password in an FTP URL:

Esempio 1. rawurlencode() example 1

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

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

Esempio 2. rawurlencode() example 2

```
<?php
echo '<a href="http://example.com/department_list_script/',
     rawurlencode('sales and marketing/Miami'), '>';
?>
```

See also [rawurldecode\(\)](#), [urldecode\(\)](#), [urlencode\(\)](#) and [RFC 1738](#).

urldecode

(PHP 3, PHP 4, PHP 5)

urldecode -- Decodes URL-encoded string

Description

string **urldecode** (string str)

Decodes any %## encoding in the given string. The decoded string is returned.

Esempio 1. urldecode() example

```
<?php
$a = explode('&', $_SERVER['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\(\)](#) and [rawurldecode\(\)](#).

urlencode

(PHP 3, PHP 4, PHP 5)

urlencode -- URL-encodes string

Description

string **urlencode** (string str)

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 RFC1738 encoding (see [rawurlencode\(\)](#)) in that for historical reasons, spaces are encoded as plus (+) signs. 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:

Esempio 1. `urlencode()` example

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

Note: Be careful about variables that may match HTML entities. Things like `&`, `©` and `£` 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\(\)](#).

Esempio 2. `urlencode()` and [htmlentities\(\)](#) example

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

See also [urldecode\(\)](#), [htmlentities\(\)](#), [rawurldecode\(\)](#) and [rawurlencode\(\)](#).