json_decode

(PHP 5 >= 5.2.0, PECL json >= 1.2.0, PHP 7)

json_decode — Decodes a JSON string

Description ¶

mixed **json_decode** (string `$json` [, bool `$assoc` = false [, int `$depth` = 512 [, int `$options` = 0]]])

Takes a JSON encoded string and converts it into a PHP variable.

Parameters ¶

`json`

The `json` string being decoded.

This function only works with UTF-8 encoded strings.

Note:

PHP implements a superset of JSON as specified in the original » RFC 7159.

`assoc`

When **`TRUE`**, returned objects will be converted into associative arrays.

`depth`

User specified recursion depth.

`options`

Bitmask of JSON decode options. Currently there are two supported options. The first is **`JSON_BIGINT_AS_STRING`** that allows casting big integers to string instead of floats which is the default. The second option is **`JSON_OBJECT_AS_ARRAY`** that has the same effect as setting `assoc` to **`TRUE`**.

Return Values ¶

Returns the value encoded in `json` in appropriate PHP type. Values *true*, *false* and *null* are returned as **`TRUE`**, **`FALSE`** and **`NULL`** respectively. **`NULL`** is returned if the `json` cannot be decoded or if the encoded data is deeper than the recursion limit.

Examples ¶

Example #1 json_decode() examples

```
<?php
$json = '{"a":1,"b":2,"c":3,"d":4,"e":5}';

var_dump(json_decode($json));
var_dump(json_decode($json, true));

?>
```

The above example will output:

```
object(stdClass)#1 (5) {
    ["a"] => int(1)
    ["b"] => int(2)
    ["c"] => int(3)
    ["d"] => int(4)
    ["e"] => int(5)
}

array(5) {
    ["a"] => int(1)
    ["b"] => int(2)
    ["c"] => int(3)
    ["d"] => int(4)
    ["e"] => int(5)
}
```

Example #2 Accessing invalid object properties

Accessing elements within an object that contain characters not permitted under PHP's naming convention (e.g. the hyphen) can be accomplished by encapsulating the element name within braces and the apostrophe.

```php
<?php

$json = '{"foo-bar": 12345}';

$obj = json_decode($json);
print $obj->{'foo-bar'}; // 12345

?>
```

Example #3 common mistakes using json_decode()

```php
<?php

// the following strings are valid JavaScript but not valid JSON

// the name and value must be enclosed in double quotes
// single quotes are not valid
$bad_json = "{ 'bar': 'baz' }";
json_decode($bad_json); // null

// the name must be enclosed in double quotes
$bad_json = '{ bar: "baz" }';
json_decode($bad_json); // null

// trailing commas are not allowed
$bad_json = '{ bar: "baz", }';
json_decode($bad_json); // null

?>
```

Example #4 `depth` errors

```php
<?php
// Encode the data.
$json = json_encode(
    array(
        1 => array(
            'English' => array(
                'One',
                'January'
            ),
            'French' => array(
                'Une',
                'Janvier'
            )
        )
    )
);

// Define the errors.
$constants = get_defined_constants(true);
$json_errors = array();
foreach ($constants["json"] as $name => $value) {
    if (!strncmp($name, "JSON_ERROR_", 11)) {
        $json_errors[$value] = $name;
    }
}

// Show the errors for different depths.
foreach (range(4, 3, -1) as $depth) {
    var_dump(json_decode($json, true, $depth));
    echo 'Last error: ', $json_errors[json_last_error()], PHP_EOL, PHP_EOL;
}
?>
```

The above example will output:

```
array(1) {
  [1]=>
  array(2) {
    ["English"]=>
    array(2) {
      [0]=>
      string(3) "One"
      [1]=>
      string(7) "January"
    }
    ["French"]=>
    array(2) {
      [0]=>
      string(3) "Une"
      [1]=>
      string(7) "Janvier"
    }
  }
}
Last error: JSON_ERROR_NONE

NULL
Last error: JSON_ERROR_DEPTH
```

Example #5 json_decode() of large integers

```php
<?php
$json = '{"number": 12345678901234567890}';

var_dump(json_decode($json));
var_dump(json_decode($json, false, 512, JSON_BIGINT_AS_STRING));

?>
```

The above example will output:

```
object(stdClass)#1 (1) {
  ["number"]=>
  float(1.2345678901235E+19)
}
object(stdClass)#1 (1) {
  ["number"]=>
  string(20) "12345678901234567890"
}
```

Notes ¶

Note:

The JSON spec is not JavaScript, but a subset of JavaScript.

Note:

In the event of a failure to decode, json_last_error() can be used to determine the exact nature of the error.

Changelog ¶

Version	Description
7.1.0	An empty JSON key ("") can be encoded to the empty object property instead of using a key with value _*empty*_.
7.0.0	Rejected RFC 7159 incompatible number formats - top level (07, 0xff, .1, -.1) and all levels ([1.], [1.e1])
7.0.0	An empty PHP string or value that after casting to string is an empty string (*NULL*, *FALSE*) results in JSON syntax error.
5.6.0	Invalid non-lowercased variants of the *true*, *false* and *null* literals are no longer accepted as valid input, and will generate warnings.
5.4.0	The `options` parameter was added.
5.3.0	Added the optional `depth`. The default recursion depth was increased from 128 to 512
5.2.3	The nesting limit was increased from 20 to 128
5.2.1	Added support for JSON decoding of basic types.

See Also ¶

- json_encode() - Returns the JSON representation of a value
- json_last_error() - Returns the last error occurred