Output Buffering Control

Introduction

The Output Control functions allow you to control when output is sent from the script. This can be useful in several different situations, especially if you need to send headers to the browser after your script has began outputting data. The Output Control functions do not affect headers sent using header() or setcookie(), only functions such as echo() and data between blocks of PHP code.

Note
When upgrading from PHP 4.1.x (and 4.2.x) to 4.3.x due to a bug in earlier versions you must ensure that *implict_flush* is *OFF* in your *php.ini*, otherwise any output with ob_start() will not be hidden from output.

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

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

Output Control configuration options

Name	Default	Changeable	Changelog
output_buffering	"0"	PHP_INI_PERDIR	
output_handler	NULL	PHP_INI_PERDIR	Available since PHP 4.0.4.
implicit_flush	"0"	PHP_INI_ALL	PHP_INI_PERDIR in PHP <= 4.2.3.

For further details and definitions of the PHP_INI_* constants, see the php.ini directives.

Here's a short explanation of the configuration directives.

`output_buffering` boolean / integer
: You can enable output buffering for all files by setting this directive to 'On'. If you wish to limit the size of the buffer to a certain size - you can use a maximum number of bytes instead of 'On', as a value for this directive (e.g., output_buffering=4096). As of PHP 4.3.5, this directive is always Off in PHP-CLI.

`output_handler` string
: You can redirect all of the output of your scripts to a function. For example, if you set output_handler to mb_output_handler(), character encoding will be transparently converted to the specified encoding. Setting any output handler automatically turns on output buffering.

Note
You cannot use both mb_output_handler() with ob_iconv_handler() and you cannot use both ob_gzhandler() and zlib.output_compression.

Note
Only built-in functions can be used with this directive. For user defined functions, use ob_start().

implicit_flush boolean
FALSE by default. Changing this to **TRUE** tells PHP to tell the output layer to flush itself automatically after every output block. This is equivalent to calling the PHP function flush() after each and every call to print() or echo() and each and every *HTML* block. When using PHP within an web environment, turning this option on has serious performance implications and is generally recommended for debugging purposes only. This value defaults to **TRUE** when operating under the *CLI SAPI*. See also ob_implicit_flush().

Resource Types

This extension has no resource types defined.

Predefined Constants

This extension has no constants defined.

Examples

Examples

Example #1 - Output Control example

```
<?php

ob_start();
echo "Hello\n";

setcookie("cookiename", "cookiedata");

ob_end_flush();

?>
```

In the above example, the output from echo() would be stored in the output buffer until ob_end_flush() was called. In the mean time, the call to setcookie() successfully stored a cookie without causing an error. (You can not normally send headers to the browser after data has already been sent.)

Output Control Functions

See Also

See also header() and setcookie().

flush

flush -- Flush the output buffer

Description

void **flush** (void)

Flushes the output buffers of PHP and whatever backend PHP is using (CGI, a web server, etc). This effectively tries to push all the output so far to the user's browser.

flush() has no effect on the buffering scheme of your web server or the browser on the client side. Thus you need to call both ob_flush() and flush() to flush the output buffers.

Several servers, especially on Win32, will still buffer the output from your script until it terminates before transmitting the results to the browser.

Server modules for Apache like mod_gzip may do buffering of their own that will cause flush() to not result in data being sent immediately to the client.

Even the browser may buffer its input before displaying it. Netscape, for example, buffers text until it receives an end-of-line or the beginning of a tag, and it won't render tables until the </table> tag of the outermost table is seen.

Some versions of Microsoft Internet Explorer will only start to display the page after they have received 256 bytes of output, so you may need to send extra whitespace before flushing to get those browsers to display the page.

Return Values

No value is returned.

ob_clean

ob_clean -- Clean (erase) the output buffer

Description

void **ob_clean** (void)

This function discards the contents of the output buffer.

This function does not destroy the output buffer like ob_end_clean() does.

Return Values

No value is returned.

See Also

- ob_flush()
- ob_end_flush()
- ob_end_clean()

ob_end_clean

ob_end_clean -- Clean (erase) the output buffer and turn off output buffering

Description

bool **ob_end_clean** (void)

This function discards the contents of the topmost output buffer and turns off this output buffering. If you want to further process the buffer's contents you have to call ob_get_contents() before ob_end_clean() as the buffer contents are discarded when ob_end_clean() is called.

Return Values

Returns **TRUE** on success or **FALSE** on failure. Reasons for failure are first that you called the function without an active buffer or that for some reason a buffer could not be deleted (possible for special buffer).

Errors/Exceptions

If the function fails it generates an **E_NOTICE**.

ChangeLog

Version	Description
4.2.0	The boolean return value was added.

Examples

The following example shows an easy way to get rid of all output buffers:

Example #2 - ob_end_clean() example

```
<?php
ob_start();
echo 'Text that won\'t get displayed.';
ob_end_clean();
?>
```

See Also

- ob_start()
- ob_get_contents()
- ob_flush()

ob_end_flush

ob_end_flush -- Flush (send) the output buffer and turn off output buffering

Description

bool **ob_end_flush** (void)

This function will send the contents of the topmost output buffer (if any) and turn this output buffer off. If you want to further process the buffer's contents you have to call ob_get_contents() before ob_end_flush() as the buffer contents are discarded after ob_end_flush() is called.

Note
This function is similar to ob_get_flush(), except that ob_get_flush() returns the buffer as a string.

Return Values

Returns **TRUE** on success or **FALSE** on failure. Reasons for failure are first that you called the function without an active buffer or that for some reason a buffer could not be deleted (possible for special buffer).

Errors/Exceptions

If the function fails it generates an **E_NOTICE**.

ChangeLog

Version	**Description**
4.2.0	The boolean return value was added.

Examples

Example #3 - ob_end_flush() example
The following example shows an easy way to flush and end all output buffers:

```
<?php
 while (@ob_end_flush());
?>
```

See Also

- ob_start()
- ob_get_contents()
- ob_get_flush()
- ob_flush()
- ob_end_clean()

ob_flush

ob_flush -- Flush (send) the output buffer

Description

void **ob_flush** (void)

This function will send the contents of the output buffer (if any). If you want to further process the buffer's contents you have to call ob_get_contents() before ob_flush() as the buffer contents are discarded after ob_flush() is called.

This function does not destroy the output buffer like ob_end_flush() does.

Return Values

No value is returned.

See Also

- ob_get_contents()
- ob_clean()
- ob_end_flush()
- ob_end_clean()

ob_get_clean

ob_get_clean -- Get current buffer contents and delete current output buffer

Description

string **ob_get_clean** (void)

Gets the current buffer contents and delete current output buffer.

ob_get_clean() essentially executes both ob_get_contents() and ob_end_clean().

Return Values

Returns the contents of the output buffer and end output buffering. If output buffering isn't active then **FALSE** is returned.

Examples

Example #4 - A simple ob_get_clean() example

```
<?php

ob_start();

echo "Hello World";

$out = ob_get_clean();
$out = strtolower($out);

var_dump($out);
?>
```

The above example will output:

```
string(11) "hello world"
```

See Also

- ob_get_contents()
- ob_start()

ob_get_contents

ob_get_contents -- Return the contents of the output buffer

Description

string **ob_get_contents** (void)

Gets the contents of the output buffer without clearing it.

Return Values

This will return the contents of the output buffer or **FALSE**, if output buffering isn't active.

Examples

Example #5 - A simple ob_get_contents() example

```
<?php

ob_start();

echo "Hello ";

$out1 = ob_get_contents();

echo "World";

$out2 = ob_get_contents();

ob_end_clean();

var_dump($out1, $out2);
?>
```

The above example will output:

```
string(6) "Hello "
string(11) "Hello World"
```

See Also

- ob_start()
- ob_get_length()

ob_get_flush

ob_get_flush -- Flush the output buffer, return it as a string and turn off output buffering

Description

string **ob_get_flush** (void)

ob_get_flush() flushes the output buffer, return it as a string and turns off output buffering.

Note
This function is similar to ob_end_flush(), except that this function returns the buffer as a string.

Return Values

Returns the output buffer or **FALSE** if no buffering is active.

Examples

Example #6 - ob_get_flush() example

```
<?php
//using output_buffering=On
print_r(ob_list_handlers());

//save buffer in a file
$buffer = ob_get_flush();
file_put_contents('buffer.txt', $buffer);

print_r(ob_list_handlers());
?>
```

The above example will output:

```
Array
(
   [0] => default output handler
)
Array
(
)
```

See Also

- ob_end_clean()
- ob_end_flush()
- ob_list_handlers()

ob_get_length

ob_get_length -- Return the length of the output buffer

Description

int **ob_get_length** (void)

This will return the length of the contents in the output buffer.

Return Values

Returns the length of the output buffer contents or **FALSE** if no buffering is active.

Examples

Example #7 - A simple ob_get_length() example

```
<?php

ob_start();

echo "Hello ";

$len1 = ob_get_length();

echo "World";

$len2 = ob_get_length();

ob_end_clean();

echo $len1 . ", ." . $len2;
?>
```

The above example will output:

```
6, 11
```

See Also

- ob_start()
- ob_get_contents()

ob_get_level

ob_get_level -- Return the nesting level of the output buffering mechanism

Description

int **ob_get_level** (void)

Returns the nesting level of the output buffering mechanism.

Return Values

Returns the level of nested output buffering handlers or zero if output buffering is not active.

See Also

- ob_start()
- ob_get_contents()

ob_get_status

ob_get_status -- Get status of output buffers

Description

array **ob_get_status** ([bool `$full_status` = FALSE])

ob_get_status() returns status information on either the top level output buffer or all active output buffer levels if *`full_status`* is set to **TRUE**.

Parameters

`full_status`
: **TRUE** to return all active output buffer levels. If **FALSE** or not set, only the top level output buffer is returned.

Return Values

If called without the *`full_status`* parameter or with *`full_status`* = **FALSE** a simple array with the following elements is returned:

```
Array
(
   [level] => 2
   [type] => 0
   [status] => 0
   [name] => URL-Rewriter
   [del] => 1
)
```

Simple ob_get_status() **results**
Key: level
Value: Output nesting level
Key: type
Value: *PHP_OUTPUT_HANDLER_INTERNAL (0)* or *PHP_OUTPUT_HANDLER_USER (1)*
Key: status
Value: One of *PHP_OUTPUT_HANDLER_START* (0), *PHP_OUTPUT_HANDLER_CONT* (1) or *PHP_OUTPUT_HANDLER_END* (2)
Key: name
Value: Name of active output handler or ' default output handler' if none is set
Key: del
Value: Erase-flag as set by ob_start()

If called with *`full_status`* = **TRUE** an array with one element for each active output buffer

level is returned. The output level is used as key of the top level array and each array element itself is another array holding status information on one active output level.

```
Array
(
   [0] => Array
       (
           [chunk_size] => 0
           [size] => 40960
           [block_size] => 10240
           [type] => 1
           [status] => 0
           [name] => default output handler
           [del] => 1
       )

   [1] => Array
       (
           [chunk_size] => 0
           [size] => 40960
           [block_size] => 10240
           [type] => 0
           [buffer_size] => 0
           [status] => 0
           [name] => URL-Rewriter
           [del] => 1
       )

)
```

The full output contains these additional elements:
Full ob_get_status() **results**
Key: chunk_size
Value: Chunk size as set by ob_start()
Key: size
Value:...
Key: blocksize
Value:...

See Also

- ob_get_level()
- ob_list_handlers()

ob_gzhandler

ob_gzhandler -- ob_start callback function to gzip output buffer

Description

string **ob_gzhandler** (string `$buffer`, int `$mode`)

ob_gzhandler() is intended to be used as a callback function for ob_start() to help facilitate sending gz-encoded data to web browsers that support compressed web pages. Before ob_gzhandler() actually sends compressed data, it determines what type of content encoding the browser will accept ("gzip", "deflate" or none at all) and will return its output accordingly. All browsers are supported since it's up to the browser to send the correct header saying that it accepts compressed web pages. If a browser doesn't support compressed pages this function returns **FALSE**.

Parameters

`buffer`

`mode`

Return Values

ChangeLog

Version	Description
4.0.5	The *`mode`* parameter was added.

Examples

Example #8 - ob_gzhandler() example

```
<?php

ob_start("ob_gzhandler");
```

```
?>
<html>
<body>
<p>This should be a compressed page.</p>
</html>
<body>
```

Notes

Note
ob_gzhandler() requires the zlib extension.

Note
You cannot use both ob_gzhandler() and zlib.output_compression. Also note that using zlib.output_compression is preferred over ob_gzhandler().

See Also

- ob_start()
- ob_end_flush()

ob_implicit_flush

ob_implicit_flush -- Turn implicit flush on/off

Description

void **ob_implicit_flush** ([int `$flag`])

ob_implicit_flush() will turn implicit flushing on or off. Implicit flushing will result in a flush operation after every output call, so that explicit calls to flush() will no longer be needed.

Parameters

`flag`
: **TRUE** to turn implicit flushing on, **FALSE** otherwise. Defaults to **TRUE**.

Return Values

No value is returned.

See Also

- flush()
- ob_start()
- ob_end_flush()

ob_list_handlers

ob_list_handlers -- List all output handlers in use

Description

array **ob_list_handlers** (void)

Lists all output handlers in use.

Return Values

This will return an array with the output handlers in use (if any). If output_buffering is enabled or an anonymous function was used with ob_start(), ob_list_handlers() will return "default output handler".

Examples

Example #9 - ob_list_handlers() example

```
<?php
//using output_buffering=On
print_r(ob_list_handlers());
ob_end_flush();

ob_start("ob_gzhandler");
print_r(ob_list_handlers());
ob_end_flush();

// anonymous functions
ob_start(create_function('$string', 'return $string;'));
print_r(ob_list_handlers());
ob_end_flush();
?>
```

The above example will output:

```
Array
(
   [0] => default output handler
)

Array
(
   [0] => ob_gzhandler
)

Array
(
   [0] => default output handler
)
```

See Also

- ob_end_clean()
- ob_end_flush()
- ob_get_flush()
- ob_start()

ob_start

ob_start -- Turn on output buffering

Description

bool **ob_start** ([callback `$output_callback` [, int `$chunk_size` [, bool `$erase`]]])

This function will turn output buffering on. While output buffering is active no output is sent from the script (other than headers), instead the output is stored in an internal buffer.

The contents of this internal buffer may be copied into a string variable using ob_get_contents(). To output what is stored in the internal buffer, use ob_end_flush(). Alternatively, ob_end_clean() will silently discard the buffer contents.

Warning
Some web servers (e.g. Apache) change the working directory of a script when calling the callback function. You can change it back by e.g. *chdir(dirname($_SERVER['SCRIPT_FILENAME']))* in the callback function.

Output buffers are stackable, that is, you may call ob_start() while another ob_start() is active. Just make sure that you call ob_end_flush() the appropriate number of times. If multiple output callback functions are active, output is being filtered sequentially through each of them in nesting order.

Parameters

`output_callback`

An optional *`output_callback`* function may be specified. This function takes a string as a parameter and should return a string. The function will be called when ob_end_flush() is called, or when the output buffer is flushed to the browser at the end of the request. When *`output_callback`* is called, it will receive the contents of the output buffer as its parameter and is expected to return a new output buffer as a result, which will be sent to the browser. If the *`output_callback`* is not a callable function, this function will return **FALSE**. If the callback function has two parameters, the second parameter is filled with a bit-field consisting of **PHP_OUTPUT_HANDLER_START**, **PHP_OUTPUT_HANDLER_CONT** and **PHP_OUTPUT_HANDLER_END**. If *`output_callback`* returns **FALSE** original input is sent to the browser. The *`output_callback`* parameter may be bypassed by passing a **NULL** value.
ob_end_clean(), ob_end_flush(), ob_clean(), ob_flush() and ob_start() may not be called from a callback function. If you call them from callback function, the behavior is undefined. If you would like to delete the contents of a buffer, return "" (a null string) from callback function. You can't even call functions using the output buffering functions like *print_r($expression, true)* or *highlight_file($filename, true)* from a callback function.

Note
In PHP 4.0.4, ob_gzhandler() was introduced to facilitate sending gz-encoded data to web browsers that support compressed web pages. ob_gzhandler() determines what type of content encoding the browser will accept and will return its output accordingly.

chunk_size

If the optional parameter *chunk_size* is passed, the buffer will be flushed after any output call which causes the buffer's length to equal or exceed *chunk_size*. Default value 0 means that the function is called only in the end, other special value 1 sets *chunk_size* to 4096.

erase

If the optional parameter *erase* is set to **FALSE**, the buffer will not be deleted until the script finishes (as of PHP 4.3.0).

Return Values

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

ChangeLog

Version	Description
4.3.2	This function was changed to return **FALSE** in case the passed *output_callback* can not be executed.

Examples

Example #10 - User defined callback function example

```
<?php

function callback($buffer)
{
 // replace all the apples with oranges
 return (str_replace("apples", "oranges", $buffer));
}

ob_start("callback");
```

```
?>
<html>
<body>
<p>It's like comparing apples to oranges.</p>
</body>
</html>
<?php

ob_end_flush();

?>
```

The above example will output:

```
<html>
<body>
<p>It's like comparing oranges to oranges.</p>
</body>
</html>
```

See Also

- ob_get_contents()
- ob_end_clean()
- ob_end_flush()
- ob_implicit_flush()
- ob_gzhandler()
- ob_iconv_handler()
- mb_output_handler()
- ob_tidyhandler()

output_add_rewrite_var

output_add_rewrite_var -- Add URL rewriter values

Description

bool **output_add_rewrite_var** (string `$name`, string `$value`)

This function adds another name/value pair to the URL rewrite mechanism. The name and value will be added to URLs (as GET parameter) and forms (as hidden input fields) the same way as the session ID when transparent URL rewriting is enabled with session.use_trans_sid. Please note that absolute URLs (http://example.com/..) aren't rewritten.

This function's behavior is controlled by the url_rewriter.tags *php.ini* parameter.

Note
Calling this function will implicitly start output buffering if it is not active already.

Parameters

`name`
The variable name.

`value`
The variable value.

Return Values

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

Examples

Example #11 - output_add_rewrite_var() example

```
<?php
output_add_rewrite_var('var', 'value');

// some links
echo '<a href="file.php">link</a>
<a href="http://example.com">link2</a>';

// a form
```

```
echo '<form action="script.php" method="post">
<input type="text" name="var2" />
</form>';

print_r(ob_list_handlers());
?>
```

The above example will output:

```
<a href="file.php?var=value">link</a>
<a href="http://example.com">link2</a>

<form action="script.php" method="post">
<input type="hidden" name="var" value="value" />
<input type="text" name="var2" />
</form>

Array
(
   [0] => URL-Rewriter
)
```

See Also

- output_reset_rewrite_vars()
- ob_flush()
- ob_list_handlers()

output_reset_rewrite_vars

output_reset_rewrite_vars -- Reset URL rewriter values

Description

bool **output_reset_rewrite_vars** (void)

This function resets the URL rewriter and removes all rewrite variables previously set by the output_add_rewrite_var() function or the session mechanism (if *session.use_trans_sid* was set on session_start()).

Return Values

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

Examples

Example #12 - output_reset_rewrite_vars() example

```
<?php
session_start();
output_add_rewrite_var('var', 'value');

echo '<a href="file.php">link</a>';
ob_flush();

output_reset_rewrite_vars();
echo '<a href="file.php">link</a>';
?>
```

The above example will output:

```
<a href="file.php?PHPSESSID=xxx&var=value">link</a>
<a href="file.php">link</a>
```

See Also

- output_add_rewrite_var()
- ob_flush()
- ob_list_handlers()
- session_start()