Variables From External Sources ¶

HTML Forms (GET and POST) ¶

When a form is submitted to a PHP script, the information from that form is automatically made available to the script. There are few ways to access this information, for example:

Example #1 A simple HTML form

```
<form action="foo.php" method="post">
    Name:  <input type="text" name="username" /><br />
    Email: <input type="text" name="email" /><br />
    <input type="submit" name="submit" value="Submit me!" />
</form>
```

As of PHP 5.4.0, there are only two ways to access data from your HTML forms. Currently available methods are listed below:

Example #2 Accessing data from a simple POST HTML form

```
<?php
echo $_POST['username'];
echo $_REQUEST['username'];
?>
```

There were some other ways of accessing user input in old PHP versions. These are listed below. See changelog at the bottom of the page for more details.

Example #3 Old methods of accessing user input

```
<?php
// WATCH OUT: these methods ARE NOT supported anymore.
// Valid ones were described above.

// Using import_request_variables() - this function has been removed in PHP 5.4.0
   import_request_variables('p', 'p_');
   echo $p_username;

// These long predefined variables were removed in PHP 5.4.0
   echo $HTTP_POST_VARS['username'];

// Using register_globals. This feature was removed in PHP 5.4.0
```

```
   echo $username;
?>
```

Using a GET form is similar except you'll use the appropriate GET predefined variable instead. GET also applies to the *QUERY_STRING* (the information after the '?' in a URL). So, for example, *http://www.example.com/test.php?id=3* contains GET data which is accessible with *$_GET['id']*. See also *$_REQUEST*.

> **Note**:
>
> Dots and spaces in variable names are converted to underscores. For example *<input name="a.b" />* becomes *$_REQUEST["a_b"]*.

PHP also understands arrays in the context of form variables (see the related faq). You may, for example, group related variables together, or use this feature to retrieve values from a multiple select input. For example, let's post a form to itself and upon submission display the data:

Example #4 More complex form variables

```
<?php
if ($_POST) {
    echo '<pre>';
    echo htmlspecialchars(print_r($_POST, true));
    echo '</pre>';
}
?>
<form action="" method="post">
    Name:  <input type="text" name="personal[name]" /><br />
    Email: <input type="text" name="personal[email]" /><br />
    Beer: <br />
    <select multiple name="beer[]">
        <option value="warthog">Warthog</option>
        <option value="guinness">Guinness</option>
        <option value="stuttgarter">Stuttgarter Schwabenbräu</option>
    </select><br />
    <input type="submit" value="submit me!" />
</form>
```

> **Note**: If an external variable name begins with a valid array syntax, trailing characters are silently ignored. For example, *<input name="foo[bar]baz">* becomes *$_REQUEST['foo']['bar']*.

IMAGE SUBMIT variable names ¶

When submitting a form, it is possible to use an image instead of the standard submit button with a tag like:

```
<input type="image" src="image.gif" name="sub" />
```

When the user clicks somewhere on the image, the accompanying form will be transmitted to the server with two additional variables, *sub_x* and *sub_y*. These contain the coordinates of the user click within the image. The experienced may note that the actual variable names sent by the browser contains a period rather than an underscore, but PHP converts the period to an underscore automatically.

HTTP Cookies ¶

PHP transparently supports HTTP cookies as defined by » RFC 6265. Cookies are a mechanism for storing data in the remote browser and thus tracking or identifying return users. You can set cookies using the setcookie() function. Cookies are part of the HTTP header, so the SetCookie function must be called before any output is sent to the browser. This is the same restriction as for the header() function. Cookie data is then available in the appropriate cookie data arrays, such as *$_COOKIE* as well as in *$_REQUEST*. See the setcookie() manual page for more details and examples.

If you wish to assign multiple values to a single cookie variable, you may assign it as an array. For example:

```
<?php
  setcookie("MyCookie[foo]", 'Testing 1', time()+3600);
  setcookie("MyCookie[bar]", 'Testing 2', time()+3600);
?>
```

That will create two separate cookies although *MyCookie* will now be a single array in your script. If you want to set just one cookie with multiple values, consider using serialize() or explode() on the value first.

Note that a cookie will replace a previous cookie by the same name in your browser unless the path or domain is different. So, for a shopping cart application you may want to keep a counter and pass this along. i.e.

Example #5 A setcookie() example

```
<?php
if (isset($_COOKIE['count'])) {
    $count = $_COOKIE['count'] + 1;
} else {
    $count = 1;
}
setcookie('count', $count, time()+3600);
setcookie("Cart[$count]", $item, time()+3600);
?>
```

Dots in incoming variable names ¶

Typically, PHP does not alter the names of variables when they are passed into a script. However, it should be noted that the dot (period, full stop) is not a valid character in a PHP variable name. For the reason, look at it:

```
<?php
$varname.ext;  /* invalid variable name */
?>
```

Now, what the parser sees is a variable named *$varname*, followed by the string concatenation operator, followed by the barestring (i.e. unquoted string which doesn't match any known key or reserved words) 'ext'. Obviously, this doesn't have the intended result.

For this reason, it is important to note that PHP will automatically replace any dots in incoming variable names with underscores.

Determining variable types ¶

Because PHP determines the types of variables and converts them (generally) as needed, it is not always obvious what type a given variable is at any one time. PHP includes several functions which find out what type a variable is, such as: gettype(), is_array(), is_float(), is_int(), is_object(), and is_string(). See also the chapter on Types.

HTTP being a text protocol, most, if not all, content that comes in Superglobal arrays, like *$_POST* and *$_GET* will remain as strings. PHP will not try to convert values to a specific type. In the example below, *$_GET["var1"]* will contain the string "null" and *$_GET["var2"]*, the string "123".

```
/index.php?var1=null&var2=123
```

Changelog ¶

Version	Description
5.4.0	Register Globals, Magic Quotes and register_long_arrays has been removed
5.3.0	Register Globals, Magic Quotes and register_long_arrays became deprecated

add a note

User Contributed Notes 30 notes

up
down
29
Anonymous ¶
11 years ago

```
The full list of field-name characters that PHP converts to _ (underscore) is the following (not just dot):
chr(32) ( ) (space)
chr(46) (.) (dot)
chr(91) ([) (open square bracket)
chr(128) - chr(159) (various)

PHP irreversibly modifies field names containing these characters in an attempt to maintain compatibility with the deprecated
register_globals feature.
```

up
down
8
yasuo_ohgaki at hotmail dot com ¶
18 years ago

```
Important:  Pay attention to the following security concerns when handling user submitted  data :

http://www.php.net/manual/en/security.registerglobals.php
http://www.php.net/manual/en/security.variables.php
```

up
down
10
krydprz at iit dot edu ¶
13 years ago

```
This post is with regards to handling forms that have more than one submit button.

Suppose we have an HTML form with a submit button specified like this:

<input type="submit" value="Delete" name="action_button">

Normally the 'value' attribute of the HTML 'input' tag (in this case "Delete") that creates the submit button can be accessed in PHP after
post like this:

<?php
$_POST['action_button'];
?>

We of course use the 'name' of the button as an index into the $_POST array.

This works fine, except when we want to pass more information with the click of this particular button.

Imagine a scenario where you're dealing with user management in some administrative interface.  You are presented with a list of user names
queried from a database and wish to add a "Delete" and "Modify" button next to each of the names in the list.  Naturally the 'value' of our
buttons in the HTML form that we want to display will be "Delete" and "Modify" since that's what we want to appear on the buttons'
faceplates.

Both buttons (Modify and Delete) will be named "action_button" since that's what we want to index the $_POST array with.  In other words, the
```

'name' of the buttons along cannot carry any uniquely identifying information if we want to process them systematically after submit. Since these buttons will exist for every user in the list, we need some further way to distinguish them, so that we know for which user one of the buttons has been pressed.

Using arrays is the way to go. Assuming that we know the unique numerical identifier of each user, such as their primary key from the database, and we DON'T wish to protect that number from the public, we can make the 'action_button' into an array and use the user's unique numerical identifier as a key in this array.

Our HTML code to display the buttons will become:

```
<input type="submit" value="Delete" name="action_button[0000000002]">
<input type="submit" value="Modify" name="action_button[0000000002]">
```

The 0000000002 is of course the unique numerical identifier for this particular user.

Then when we handle this form in PHP we need to do the following to extract both the 'value' of the button ("Delete" or "Modify") and the unique numerical identifier of the user we wish to affect (0000000002 in this case). The following will print either "Modify" or "Delete", as well as the unique number of the user:

```
<?php
$submitted_array = array_keys($_POST['action_button']);
echo ($_POST['action_button'][$submitted_array[0]] . " " . $submitted_array[0]);
?>
```

$submitted_array[0] carries the 0000000002.
When we index that into the $_POST['action_button'], like we did above, we will extract the string that was used as 'value' in the HTML code 'input' tag that created this button.

If we wish to protect the unique numerical identifier, we must use some other uniquely identifying attribute of each user. Possibly that attribute should be encrypted when output into the form for greater security.

Enjoy!
up
down
6
tmk-php at infeline dot org ¶
14 years ago
To handle forms with or without [] you can do something like this:

```
<?php
    function repairPost($data) {
        // combine rawpost and $_POST ($data) to rebuild broken arrays in $_POST
        $rawpost = "&".file_get_contents("php://input");
        while(list($key,$value)= each($data)) {
            $pos = preg_match_all("/&".$key."=([^&]*)/i",$rawpost, $regs, PREG_PATTERN_ORDER);
            if((!is_array($value)) && ($pos > 1)) {
                $qform[$key] = array();
                for($i = 0; $i < $pos; $i++) {
                    $qform[$key][$i] = urldecode($regs[1][$i]);
                }
            } else {
                $qform[$key] = $value;
            }
        }
        return $qform;
    }

    // --- MAIN

    $_POST = repairPost($_POST);
?>
```

The function will check every field in the $_POST with the raw post data and rebuild the arrays that got lost.
up
down
5
Anonymous ¶
5 years ago
From HTML 5.1 Draft:
http://www.w3.org/html/wg/drafts/html/master/forms.html#naming-form-controls:-the-name-attribute

The name content attribute gives the name of the form control, as used in form submission and in the form element's elements object. If the attribute is specified, its value must not be the empty string.
Any non-empty value for name is allowed.

So use the format like this <select multiple name="beer[]"> is still in the HTML 5 standard.
up
down
4
lennynyktyk at yahoo dot com ¶
14 years ago
When dealing with multiple select boxes and the name=some_name[] so that PHP will understand that is needs to interpet the input as an array an not as a single value. If you want to access this in Javascript you should assign an id attribute to the select box as well as the name

attribute. Then proceed to use the id attribute in Javascript to reference the select box and the name attribute to reference the select box in PHP.
Example

```
<select multiple id="select_id" name="select_name[]">
....

</select>

<?PHP
    echo $select_name[0];
?>

<script language="javascript">
  document.forms[0].select_id.options[0].selected = true;
</script>
```

I hope you get the idea
up
down
3
walf ¶
7 years ago
WARNING! replacement of spaces and dots does not occur in array keys.

E.g. If you have
`<input name="a. b[x. y]" value="foo" />`

```
var_dump($_POST);
gives
array(1) {
  ["a__b"]=>
  array(1) {
    ["x. y"]=>
    string(3) "foo"
  }
}
```

up
down
3
Anonymous ¶
16 years ago
"...the dot (period, full stop) is not a valid character in a PHP variable name."

That's not completely correct, consider this example:

```
$GLOBALS['foo.bar'] = 'baz';
echo ${'foo.bar'};
```

This will output baz as expected.
up
down
2
vb at bertola dot eu dot org ¶
16 years ago
For what I understand, since PHP 4.3 it is possible to access the content of a POST request (or other methods as well) as an input stream named php://input, example:

```
readfile("php://input");
[to display it]

or

$fp = fopen("php://input", "r");
[to open it and then do whatever you want]
```

This is very useful to access the content of POST requests which actually have a content (and not just variable-value couples, which appear in $_POST).

This substitutes the old $HTTP_RAW_POST_DATA variable available in some of the previous 4.x versions. It is available for other upload methods different from POST too, but it is not available for POSTs with multipart/form-data content type, since the file upload handler has already taken care of the content in that case.
up
down
1
anisgazis at gmail dot com ¶
2 months ago
Dots , spaces and [in variable names are converted to underscores. For example
`<input name="a.b" /> becomes $_REQUEST["a_b"].`
`<input name="a b" /> becomes $_REQUEST["a_b"].`
`<input name="a[b" /> becomes $_REQUEST["a_b"].`
`<input name="a]b" /> becomes $_REQUEST["a]b"].`
`<input name="a-b" /> becomes $_REQUEST["a-b"].`
`<input name="a/b" /> becomes $_REQUEST["a/b"].`

```
<input name="a\b" /> becomes $_REQUEST["a\b"].
<input name="a,b" /> becomes $_REQUEST["a,b"].
```

up
down
0
t.montg AT gmail DOT com ¶
11 years ago

For anyone else having trouble figuring out how to access values in a SELECT element from a POST or GET form, you can't set the "id" attribute to the same thing as your "name" attribute. i.e. don't do this:

```
<?php
  //Not so good
  <select multiple="multiple" id="selectElem" name="selectElem[]">
     <option value="ham">Ham</option>
     <option value="cheese">Cheese</option>
     <option value="hamcheese">Ham and Cheese</option>
  </select>
?>
```

If you do the above, the variable $_POST['selectElem'] will not be set. Instead, either change the id or name attribute so that they are dissimilar. i.e. do this:

```
<?php
  //So good (notice the new "id" value)
  <select multiple="multiple" id="selectElemId" name="selectElem[]">
     <option value="ham">Ham</option>
     <option value="cheese">Cheese</option>
     <option value="hamcheese">Ham and Cheese</option>
  </select>
?>
```

Then you can access the value(s) of the SELECT element through the array $_POST['selectElem'][] or $_GET['selectElem'][]. It took me quite some time to figure out the problem.

up
down
-1
a at b dot c dot de ¶
17 years ago

As far as whether or not "[]" in name attributes goes, The HTML4.01 specification only requires that it be a case-insensitive CDATA token, which can quite happily include "[]". Leading and trailing whitespace may be trimmed and shouldn't be used.

It is the id= attribute which is restricted, to a case-sensitive NAME token (not to be confused with a name= attribute).

up
down
-3
jlratwil at yahoo dot com ¶
14 years ago

To get multiple selected (with "multiple") lists in <select> tag, make sure that the "name" attribute is added to braces, like this:

```
<select multiple="multiple" name="users[]">
     <option value="foo">Foo</option>
     <option value="bar">Bar</option>
</select>
```

When submitted to PHP file (assume that you have a complete form) it will return an array of strings. Otherwise, it will just return the last element of the <select> tag you selected.

up
down
-4
vierubino dot r3m0oFdisB1T at gmail dot com ¶
11 years ago

When you are using checkboxes to submit multiple choices, there is no need to use the complex method further down the page where you assign a unique name to each checkbox.

Instead, just name each checkbox as the same array, e.g.:

```
<input type="checkbox" name="items[]" value="foo" />
<input type="checkbox" name="items[]" value="bar" />
<input type="checkbox" name="items[]" value="baz" />
```

This way your $_POST["items"] variable will return as an array containing all and only the checkboxes that were clicked on.

up
down
-4
kevinrlat nospam dot ccs dot neu dot edu ¶
15 years ago

if you use an array of checkboxes to submit info to a database or what have you, be careful of the case when no boxes are checked. for example:

```
<form method="post">
<input type="checkbox" name="checkstuff[]" value="0">
<input type="checkbox" name="checkstuff[]" value="1">
<input type="checkbox" name="checkstuff[]" value="2">
```

```
. . .

</form>
```

if these are submitted and none are checked, the $_POST['checkstuff'] variable will not contain an empty array, but a NULL value. this bothered me when trying to implode() the values of my checkboxes to insert into a database, i got a warning saying the 2nd argument was the wrong type.

hope this helps!
-kevin

up
down
-4

carl_steinhilber at NOSPAMmentor dot com ¶
17 years ago

A group of identically-named checkbox form elements returning an array is a pretty standard feature of HTML forms. It would seem that, if the only way to get it to work is a non-HTML-standard-compliant workaround, it's a problem with PHP.

Since the array is passed in the header in a post, or the URL in a get, it's the PHP interpretation of those values that's failing.

up
down
-5

Murat TASARSU ¶
14 years ago

if you want your multiple select returned variable in comma seperated form you can use this. hope that helps. regards...

```
$myvariable
   Array ( [0] => one [1] => two [2] => three )
turns into
   one,two,three

<?php
$myvariable="";
$myseperator="";
foreach ( $_POST["myvariable"] as $v) {
if (!isset($nofirstcomma)) $nofirstcomma=0; else $myseperator=",";
$myvariable = $myvariable.$myseperator.$v;
}
echo $myvariable;
?>
```

up
down
-5

un shift at yahoo dot com ¶
16 years ago

This function takes a recurring form item from php://input and loads it into an array - useful for javascript/dom incompatibility with form_input_item[] names for checkboxes, multiple selects, etc. The fread maxes out at 100k on this one. I guess a more portable option would be pulling in ini_get('post_max_size') and converting it to an integer.

```
<?php
function multi_post_item($input_item_name) {
     $array_output = array();
     $in_handle = fopen("php://input", "r");
     $raw_input_items = split("&", urldecode(fread($in_handle, 100000)));
     foreach ($raw_input_items as $input_item) {
            // split this item into name/value pair
            $item = split("=", $input_item);
            // form item name
            $item_name = $item[0];
            // form item value
            $item_value = $item[1];
            if ($item_name == $input_item_name) {
                   $array_output[] = $item_value;
            }
     }
     return $array_output;
}
?>
```

up
down
-1

fabian dot picone at gmail dot com ¶
6 months ago

"That will create two separate cookies although MyCookie will now be a single array in your script. If you want to set just one cookie with multiple values, consider using serialize() or explode() on the value first."

explode should be implode in this sentence.

up
down
-5

tim at timpauly dot com ¶
13 years ago

This code module can be added to every form using require_once().
It will process any and all form data, prepending each variable with
a unique identifier (so you know which method was used to get the data).

My coding could be neater, but this sure makes processing forms much easier!

```
<?php
// ---------------------------------------------------------------
// Basic Data PHP module. This module captures all GET, POST
// and COOKIE data and processes it into variables.
// Coded April, 2005 by Timothy J. Pauly
// ---------------------------------------------------------------
//
// coo_ is prepended to each cookie variable
// get_ is prepended to each GET variable
// pos_ is prepended to each POST variable
// ses_ is prepended to each SESSION variable
// ser_ is prepended to each SERVER variable

session_start(); // initialize session data
$ArrayList = array("_POST", "_GET", "_SESSION", "_COOKIE", "_SERVER"); // create an array of the autoglobal arrays
// we want to process

foreach($ArrayList as $gblArray) // process each array in the array list
{
   $prefx = strtolower(substr($gblArray,1,3))."_"; // derive the prepend string
// from the autoglobal type name
   $tmpArray = $$gblArray;
   $keys = array_keys($tmpArray); // extract the keys from the array being processed
   foreach($keys as $key) // process each key
    {

    $arcnt = count($tmpArray[$key]);

    if ($arcnt > 1) // Break down passed arrays and
// process each element seperately
    {
      $lcount = 0;
      foreach ($tmpArray[$key] as $dval)
        {
           $prkey = $prefx.$key; // create a new key string
// with the prepend string added
           $prdata['$prkey'] = $dval; // this step could be eliminated
           ${$prkey}[$lcount] = $prdata['$prkey']; //create new key and insert the data
           $lcount++;
        }

        } else { // process passed single variables

                $prkey = $prefx.$key; // create a new key string
// with the prepend string added
                $prdata['$prkey'] = $tmpArray[$key]; // insert the data from
// the old array into the new one
                $$prkey = $prdata['$prkey']; // create the newly named
// (prepended) key pair using variable variables :-)


               }
    }
}

// ---------------------------------------------------------------
?>
```

up
down
-7
POSTer ¶
9 years ago

Here's a simple function to give you an uncorrupted version of $_POST:

```
<?php
// Function to fix up PHP's messing up POST input containing dots, etc.
function getRealPOST() {
    $pairs = explode("&", file_get_contents("php://input"));
    $vars = array();
    foreach ($pairs as $pair) {
        $nv = explode("=", $pair);
        $name = urldecode($nv[0]);
        $value = urldecode($nv[1]);
        $vars[$name] = $value;
    }
    return $vars;
```

```
}
?>
```
up
down
-6
ch1902uk at hotmail dot com ¶
13 years ago
Regarding image input buttons, above where it says:

"When the user clicks somewhere on the image, the accompanying form will be transmitted to the server with two *additional* variables, sub_x and sub_y. These contain the coordinates of the user click within the image."

This is the case with Firefox (and probably other standards browsers), however my experience with Internet Explorer is that when image inputs are clicked, they only submit the location of the click on the button and *not* the name of the input.

So if you have a form to move/delete entries like this

```
entry[]  [delete_0] [up_0] [down_0]
entry[]   [delete_1] [up_1] [down_1]
entry[]   [delete_2] [up_2] [down_2]
```

Then submitting the form in firefox will give you post variables such as

```
<?php
   $_POST['delete_2'];   // "Delete" - button value
   $_POST['delete_2_x'];   // 23 - x coord
   $_POST['delete_2_y'];   // 3 - y coord
?>
```

In IE you only get

```
<?php
   $_POST['delete_2_x'];   // 23 - x coord
   $_POST['delete_2_y'];   // 3 - y coord
?>
```

So if you are checking for what button was clicked do something like this

```
<?php
   for ($i = 0; $i < count($_POST['entry']); $i++)
   {
      if (isset($_POST['delete_' . $i . '_x']))
      {
         // do delete
      }
   }
?>
```
up
down
-2
Sadik_quake2003 at mail dot ru ¶
1 year ago
```
<form method="post">
    <select name="selecter">
        <option>one</option>
        <option>two</option>
        <option>three</option>
    </select><br />
    <input type="submit" value="send" />
</form>

<?=$_POST["selecter"];?>
```

If we change first option, that result will be: one (important: not null !)
up
down
-4
keli at kmdsz dot ro ¶
16 years ago
image type inputs apparently return their "value" argument from Mozilla, but not from IEXplorer... :(

example:

```
<input type="image" name="sb" value="first" src="first.jpg">
```

using a mozilla will give you
  $sb="first" AND $sb_x, $sb_y ... whereas from IE there's just no $sb. :(

[this in short form, as I'm still using trackvars :)]
up
down
-5
hjncom at hjncom dot net ¶

16 years ago
I think '[' and ']' are valid characters for name attributes.

http://www.w3.org/TR/html401/interact/forms.html#h-17.4
-> InputType of 'name' attribute is 'CDATA'(not 'NAME' type)

http://www.w3.org/TR/html401/types.html#h-6.2
-> about CDATA('name' attribute is not 'NAME' type!)
...CDATA is a sequence of characters from the document character set and may include character entities...

http://www.w3.org/TR/html401/sgml/entities.html
--> about Character entity references in HTML 4
([- &#91,] - &#93)

up
down
-8

darren at sullivan dot net ¶
15 years ago
This function is a simple solution for getting the array of selectes from a checkbox list or a dropdown list out of the Querry String. I took an example posted earlier and simplified it.

```
<?php
function multi_post_item($repeatedString) {
    // Gets the specified array of multiple selects and/or
    // checkboxes from the Query String
    $ArrayOfItems = array();
    $raw_input_items = split("&", $_SERVER["QUERY_STRING"]);
    foreach ($raw_input_items as $input_item) {
        $itemPair = split("=", $input_item);
        if ($itemPair[0] == $repeatedString) {
            $ArrayOfItems[] = $itemPair[1];
        }
    }
    return $ArrayOfItems;
}
?>
```

Use the name of the field as the agrument. Example:

```
<?php
$Order = $_GET['Order'];
$Name = $_GET['Name'];
$States = multi_post_item('States');
$Products = multi_post_item('Products');
?>
```

Be sure to check for NULL if there are no selections or boxes checked.

up
down
-7

mattij at nitro fi no at no dot no ¶
14 years ago
If you try to refer or pass HTML-form data which has arrays with javascript remember that you should point to that array like this

```
<script type="text/javascript">
    window.opener.document.forms[0]["to[where][we][point]"];
</script>
```

up
down
-10

arjini at mac dot com ¶
15 years ago
When dealing with form inputs named_like_this[5] and javascript, instead of trying to get PHP to do something fancy as mentioned below, just try this on the javascript side of things:

```
<form name="myForm">

<script>
my_fancy_input_name = 'array_of_things[1]';
/* now just refer to it like this in the dom tree

document[myForm][my_fancy_input_name].value

etc*/
</script>

<input type="text" name="array_of_things[1]" value="1"/>
</form>
```

No fancy PHP, in fact, you shouldn't need to change your PHP at all.

up
down
-8

jim at jamesdavis dot it ¶
15 years ago

```
How to pass a numerically indexed array.
This is the part inside the form. Notice that the name is not 'english[$r]' which you would normally write, but 'english[]'. PHP adds the
index when it receives the post and it starts at 0.

<?php

for ($r=0; $r <= count($english)-1; $r++){
         echo "<TEXTAREA NAME='english[]'>".$english[$r]."</TEXTAREA>";

}
?>
<?php

And this will get it out at the other end
function retrieve_english(){
    for ($r=0; $r <= count($_POST['english'])-1; $r++){
        echo $_POST['english'][$r]."<BR>";
    }
}
?>

Keys are useful but so are numerical indices!
Cheers everyone
```

up
down
-13
Sadik_quake2003 at mail dot ru ¶
1 year ago

```
<form method="post">
    <select name="list">
        <option>one</option>
        <option>two</option>
        <option>three</option>
    </select><br />
    <input type="submit" value="send" />
</form>

<?=$_POST["list"];?>

If we change first option, that result will be: one (important: not null !)
```

add a note

- Copyright © 2001-2019 The PHP Group
- My PHP.net
- Contact
- Other PHP.net sites
- Privacy policy