php

PHPKonf: Istanbul PHP Conference 2017

Getting Started
- Introduction
- A simple tutorial

Language Reference
- Basic syntax
- Types
- Variables
- Constants
- Expressions
- Operators
- Control Structures
- Functions
- Classes and Objects
- Namespaces
- Errors
- Exceptions
- Generators
- References Explained
- Predefined Variables
- Predefined Exceptions
- Predefined Interfaces and Classes
- Context options and parameters
- Supported Protocols and Wrappers

Security
- Introduction
- General considerations
- Installed as CGI binary
- Installed as an Apache module
- Session Security
- Filesystem Security
- Database Security
- Error Reporting
- Using Register Globals
- User Submitted Data
- Magic Quotes
- Hiding PHP
- Keeping Current

Features
- HTTP authentication with PHP
- Cookies
- Sessions

- Dealing with XForms
- Handling file uploads
- Using remote files
- Connection handling
- Persistent Database Connections
- Safe Mode
- Command line usage
- Garbage Collection
- DTrace Dynamic Tracing

Function Reference

- Affecting PHP's Behaviour
- Audio Formats Manipulation
- Authentication Services
- Command Line Specific Extensions
- Compression and Archive Extensions
- Credit Card Processing
- Cryptography Extensions
- Database Extensions
- Date and Time Related Extensions
- File System Related Extensions
- Human Language and Character Encoding Support
- Image Processing and Generation
- Mail Related Extensions
- Mathematical Extensions
- Non-Text MIME Output
- Process Control Extensions
- Other Basic Extensions
- Other Services
- Search Engine Extensions
- Server Specific Extensions
- Session Extensions
- Text Processing
- Variable and Type Related Extensions
- Web Services
- Windows Only Extensions
- XML Manipulation
- GUI Extensions

Keyboard Shortcuts

?
: This help

j
: Next menu item

k
: Previous menu item

g p
: Previous man page

g n
: Next man page

G
: Scroll to bottom

g g
: Scroll to top

PDOStatement::bindParam

(PHP 5 >= 5.1.0, PHP 7, PECL pdo >= 0.1.0)

PDOStatement::bindParam — Binds a parameter to the specified variable name

Description ¶

public bool **PDOStatement::bindParam** (mixed `$parameter` , mixed `&$variable` [, int `$data_type` = PDO::PARAM_STR [, int `$length` [, mixed `$driver_options`]]])

Binds a PHP variable to a corresponding named or question mark placeholder in the SQL statement that was used to prepare the statement. Unlike PDOStatement::bindValue(), the variable is bound as a reference and will only be evaluated at the time that PDOStatement::execute() is called.

Most parameters are input parameters, that is, parameters that are used in a read-only fashion to build up the query. Some drivers support the invocation of stored procedures that return data as output parameters, and some also as input/output parameters that both send in data and are updated to receive it.

Parameters ¶

`parameter`

Parameter identifier. For a prepared statement using named placeholders, this will be a parameter name of the form *:name*. For a prepared statement using question mark placeholders, this will be the 1-indexed position of the parameter.

`variable`

Name of the PHP variable to bind to the SQL statement parameter.

`data_type`

Explicit data type for the parameter using the *PDO::PARAM_** constants. To return an INOUT parameter from a stored procedure, use the bitwise OR operator to set the PDO::PARAM_INPUT_OUTPUT bits for the `data_type` parameter.

`length`

Length of the data type. To indicate that a parameter is an OUT parameter from a stored procedure, you must explicitly set the length.

`driver_options`

Return Values ¶

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

Examples ¶

Example #1 Execute a prepared statement with named placeholders

```
<?php
/* Execute a prepared statement by binding PHP variables */
$calories = 150;
$colour = 'red';
$sth = $dbh->prepare('SELECT name, colour, calories
    FROM fruit
    WHERE calories < :calories AND colour = :colour');
$sth->bindParam(':calories', $calories, PDO::PARAM_INT);
$sth->bindParam(':colour', $colour, PDO::PARAM_STR, 12);
$sth->execute();
?>
```

Example #2 Execute a prepared statement with question mark placeholders

```
<?php
/* Execute a prepared statement by binding PHP variables */
$calories = 150;
$colour = 'red';
$sth = $dbh->prepare('SELECT name, colour, calories
    FROM fruit
    WHERE calories < ? AND colour = ?');
$sth->bindParam(1, $calories, PDO::PARAM_INT);
$sth->bindParam(2, $colour, PDO::PARAM_STR, 12);
$sth->execute();
?>
```

Example #3 Call a stored procedure with an INOUT parameter

```
<?php
/* Call a stored procedure with an INOUT parameter */
$colour = 'red';
$sth = $dbh->prepare('CALL puree_fruit(?)');
$sth->bindParam(1, $colour, PDO::PARAM_STR|PDO::PARAM_INPUT_OUTPUT, 12);
```

```
$sth->execute();
print("After pureeing fruit, the colour is: $colour");
?>
```

See Also ¶

- PDO::prepare() - Prepares a statement for execution and returns a statement object
- PDOStatement::execute() - Executes a prepared statement
- PDOStatement::bindValue() - Binds a value to a parameter

add a note

User Contributed Notes 22 notes

up
down
100
atrandafirc at yahoo dot com ¶
6 years ago

```
I know this has been said before but I'll write a note on it too because I think it's important to
keep in mind:

If you use PDO bindParam to do a search with a LIKE condition you cannot put the percentages and
quotes to the param placeholder '%:keyword%'.

This is WRONG:
"SELECT * FROM `users` WHERE `firstname` LIKE '%:keyword%'";

The CORRECT solution is to leave clean the placeholder like this:
"SELECT * FROM `users` WHERE `firstname` LIKE :keyword";

And then add the percentages to the php variable where you store the keyword:
$keyword = "%".$keyword."%";

And finally the quotes will be automatically added by PDO when executing the query so you don't have
to worry about them.

So the full example would be:
<?php
// Get the keyword from query string
$keyword = $_GET['keyword'];
// Prepare the command
$sth = $dbh->prepare('SELECT * FROM `users` WHERE `firstname` LIKE :keyword');
// Put the percentage sing on the keyword
$keyword = "%".$keyword."%";
// Bind the parameter
$sth->bindParam(':keyword', $keyword, PDO::PARAM_STR);
?>
```

up
down
98
Vili ¶
6 years ago

```
This works ($val by reference):
<?php
foreach ($params as $key => &$val) {
    $sth->bindParam($key, $val);
}
?>

This will fail ($val by value, because bindParam needs &$variable):
<?php
foreach ($params as $key => $val) {
    $sth->bindParam($key, $val);
}
?>
```

up
down
3
pegas1981 at yandex dot ru ¶
3 years ago

```
http://technet.microsoft.com/en-us/library/ff628166(v=sql.105).aspx

When binding null data to server columns of type varbinary, binary, or varbinary(max) you should
specify binary encoding (PDO::SQLSRV_ENCODING_BINARY) using the $driver_options. See Constants for
more information about encoding constants.
Support for PDO was added in version 2.0 of the Microsoft Drivers for PHP for SQL Server.

<?php
$db = new PDO('sqlsrv:server=SQLSERVERNAME;Database=own_exchange', 'user', 'password');
$sql = "INSERT INTO dbo.files(file_name, file_source) VALUES(:file_name, :file_source)";
$stmt = $db->prepare($sql);
$stmt->bindParam(":file_name", $files->name, PDO::PARAM_STR);
$stmt->bindParam(":file_source", file_get_contents($files->tempName), PDO::PARAM_LOB, 0,
PDO::SQLSRV_ENCODING_BINARY);
$stmt->execute();
?>
```

up
down
31
Steve M ¶
7 years ago

```
Note that when using PDOStatement::bindParam an integer is changed to a string value upon
PDOStatement::execute(). (Tested with MySQL).

This can cause problems when trying to compare values using the === operator.

Example:
<?php
$active = 1;
var_dump($active);
$ps->bindParam(":active", $active, PDO::PARAM_INT);
var_dump($active);
$ps->execute();
var_dump($active);
if ($active === 1) {
    // do something here
    // note: this will fail since $active is now "1"
}
?>

results in:
int(1)
int(1)
string(1) "1"
```

up
down
9
cyrylas at gmail dot com ¶
6 years ago

```
Please note, that PDO format numbers according to current locale. So if, locale set number format to
something else, that standard that query WILL NOT work properly.

For example:
in Polish locale (pl_PL) proper decimal separator is coma (","), so: 123,45, not 123.45. If we try
bind 123.45 to the query, we will end up with coma in the query.

<?php
setlocale(LC_ALL, 'pl_PL');
$sth = $dbh->prepare('SELECT name FROM products WHERE price < :price');
$sth->bindParam(':price', 123.45, PDO::PARAM_STR);
$sth->execute();
// result:
// SELECT name FROM products WHERE price < '123,45';
?>
```

up
down
3
Ejaz Ansari ¶
10 months ago

```
For those who are confused on insert query using PDO-bindparam:

$sql = $db->prepare("INSERT INTO db_fruit (id, type, colour) VALUES (? ,? ,?)");

$sql->bindParam(1, $newId);
$sql->bindParam(2, $name);
$sql->bindParam(3, $colour);
$sql->execute();
```

up
down
8
khkiley at adamsautomation dot com ¶
4 years ago

```
SQL Server 2008 R2

If this was in the documentation, I didn't stumble across it. When using bound output parameters with
a stored procedure, the output parameters are updated AFTER the LAST rowset has been processed.

If your stored procedure does not return any rowsets (no SELECT statements) then you are set, your
output parameters will be ready as soon as the stored procedure is processed.
```

```
Otherwise you need to process the rows, and then:
<?php $stmt->nextRowset(); ?>

Once that is done for each returning rowset you will have access to the output parameters.
```

up
down
5
dhammari at q90 dot com ¶
7 years ago

```
There seems to be some confusion about whether you can bind a single value to multiple identical
placeholders. For example:

$sql = "SELECT * FROM user WHERE is_admin = :myValue AND is_deleted = :myValue ";

$params = array("myValue" => "0");

Some users have reported that attempting to bind a single parameter to multiple placeholders yields a
parameter mismatch error in PHP version 5.2.0 and earlier. Starting with version 5.2.1, however, this
seems to work just fine.

For details, see bug report 40417:
http://bugs.php.net/bug.php?id=40417
```

up
down
5
Anonymous ¶
10 years ago

```
A caution for those using bindParam() on a placeholder in a
LIKE '%...%' clause, the following code will likely not work:

<?php
$q = "SELECT id, name FROM test WHERE name like '%:foo%'";
$s = "carrot";
$sth = $dbh->prepare($q);
$sth->bindParam(':foo', $s);
$sth->execute();
?>

What is needed is something like the following:

<?php
$s = "%$s%";
$sth->bindParam(':foo', $s);
?>

This should work. Tested against mysql 4.1, PHP 5.1.3.
```

up
down
4
Mike Robinson ¶
3 years ago

```
Note that with bindParam the second parameter is passed by reference. This means that the following
will produce a warning if E_STRICT is enabled:
```

```
<?php
$stmt->bindParam('type', $object->getType());

// Strict Standards: Only variables should be passed by reference in /path/to/file.php on line 123
?>
```

If the second parameter is not an actual variable, either set the result of $object->getType(); to a variable and use that variable in bindParam or use bindValue instead.

up
down
6
flannell ¶
4 years ago

Spent all day banging my head against a brick wall.

Tried to use INOUT or OUT and getting the return variable into PHP using Mysql v5.5.16 on XAMPP.

"MySQL doesn't supporting binding output parameters via its C API. You must use SQL level variables:"

```
<?php
$stm = $db->prepare("CALL sp_mysp(:Name, :Email, @sp_result)");

$outputArray = $db->query("select @sp_result")->fetch(PDO::FETCH_ASSOC);
?>
```

So the 'workaround' for Mysql and PDO is to use two SQL calls.

Hope this helps someone.

up
down
1
willie at spenlen dot com ¶
9 years ago

If you're using the MySQL driver and have a stored procedure with an OUT or INOUT parameter, you can't (currently) use bindValue(). See http://bugs.php.net/bug.php?id=35935 for a workaround.

up
down
2
gilhildebrand at gmail dot com ¶
1 year ago

MySQL will return an error if a named placeholder has a hyphen in it:
UPDATE wardrobe SET `T-Shirt`=:T-SHIRT WHERE id=:id

Will return the following error: PDOException' with message 'SQLSTATE[HY093]: Invalid parameter number: parameter was not defined'

To resolve, just remove hyphens from named placeholders:
UPDATE wardrobe SET `T-Shirt`=:TSHIRT WHERE id=:id

up
down
3
Filofox ¶
10 years ago

```
Do not try to use the same named parameter twice in a single SQL statement, for example

<?php
$sql = 'SELECT * FROM some_table WHERE  some_value > :value OR some_value < :value';
$stmt = $dbh->prepare($sql);
$stmt->execute( array( ':value' => 3 ) );
?>

...this will return no rows and no error -- you must use each parameter once and only once.
Apparently this is expected behavior (according to this bug report: http://bugs.php.net/bug.php?
id=33886)  because of portability issues.
```

up
down
1
jeffwa+php at gmail dot com ¶
9 years ago

```
Took me forever to find this elsewhere in the notes in the manual, so I'd thought I'd put this tidbit
here to help others in the future.

When using a LIKE search in MySQL along with a prepared statement, the *value* must have the
appropriate parentheses attached before the bindParam() statement as such:

<?php
$dbc = $GLOBALS['dbc'];
$sql = "SELECT * FROM `tbl_name` WHERE tbl_col LIKE ?";
$stmt = $dbc->prepare($sql);

$value = "%{$value}%";
$stmt->bindParam($i, $value, PDO::PARAM_STR);
?>

Trying to use
<?php
$stmt->bindParam($i, "%{$value}%", PDO::PARAM_STR);
?>

will fail.
```

up
down
1
Ofir Attia ¶
2 years ago

```
/*
   method for pdo class connection, you can add your cases by    yourself and use it.
*/
class Conn{
....
....
private $stmt;
public function bind($parameter, $value, $var_type = null){
        if (is_null($var_type)) {
            switch (true) {
                                case is_bool($value):
                    $var_type = PDO::PARAM_BOOL;
```

```
                break;
            case is_int($value):
                $var_type = PDO::PARAM_INT;
                break;
            case is_null($value):
                $var_type = PDO::PARAM_NULL;
                break;
            default:
                $var_type = PDO::PARAM_STR;
        }
    }
    $this->stmt->bindValue($parameter, $value, $var_type);
}
```

up
down
0
heather dot begazo at gmail dot com ¶
1 year ago

```
The documentation says this about the length parameter for bindParam:

"To indicate that a parameter is an OUT parameter from a stored procedure, you must explicitly set
the length. "

For db2, I found that setting the length for the "INPUT_OUTPUT" parameters causes a problem for
varchar parameters that are input parameters.  The problem I found is that the stored procedure was
called, but varchar input parameters were set to null inside my stored procedure and as a result, the
stored procedure could not work properly.

Here is the signature for my stored procedure:

CREATE OR REPLACE PROCEDURE MY_SCHEMA_NAME.MY_STORED_PROCEDURE_NAME ( IN RUN_ID INTEGER,IN
V_SCHEMA_NAME VARCHAR(128),
  OUT out_rc INTEGER,OUT out_err_message VARCHAR(100),OUT out_sqlstate CHAR(5) ,OUT out_sqlcode INT)

Here is the php code that works:

$command = "Call MY_SCHEMA_NAME.MY_STORED_PROCEDURE_NAME (?,?,?,?,?,?,?)";
$stmt = $this->GuestDb->prepare($command);
$stmt->bindParam(1, $RUN_ID, PDO::PARAM_INT);
$stmt->bindParam(2, $V_SCHEMA_NAME, PDO::PARAM_STR);
$stmt->bindParam(3, $V_TABNAME, PDO::PARAM_STR);
$stmt->bindParam(4, $out_rc, PDO::PARAM_INT|PDO::PARAM_INPUT_OUTPUT);
$stmt->bindParam(5, $out_err_message, PDO::PARAM_STR|PDO::PARAM_INPUT_OUTPUT);
$stmt->bindParam(6, $out_sqlstate, PDO::PARAM_STR|PDO::PARAM_INPUT_OUTPUT);
$stmt->bindParam(7, $out_sqlcode, PDO::PARAM_INT|PDO::PARAM_INPUT_OUTPUT);

Here is the php code that does not work:

$command = "Call MY_SCHEMA_NAME.MY_STORED_PROCEDURE_NAME (?,?,?,?,?,?,?)";
$stmt = $this->GuestDb->prepare($command);
$stmt->bindParam(1, $RUN_ID, PDO::PARAM_INT,12);
$stmt->bindParam(2, $V_SCHEMA_NAME, PDO::PARAM_STR,128);
$stmt->bindParam(3, $V_TABNAME, PDO::PARAM_STR,100);
$stmt->bindParam(4, $out_rc, PDO::PARAM_INT|PDO::PARAM_INPUT_OUTPUT,12);
```

```
$stmt->bindParam(5, $out_err_message, PDO::PARAM_STR|PDO::PARAM_INPUT_OUTPUT,100);
$stmt->bindParam(6, $out_sqlstate, PDO::PARAM_STR|PDO::PARAM_INPUT_OUTPUT,6);
$stmt->bindParam(7, $out_sqlcode, PDO::PARAM_INT|PDO::PARAM_INPUT_OUTPUT,12);
```

up
down
0
Vladimir Kovpak ¶
1 year ago

```
<?php
/**
* Bind bit value:
*/

$sql = 'SELECT * FROM myTable WHERE level & ?';
$sth = \App::pdo()->prepare($sql);
$sth->bindValue(1, 0b0101, \PDO::PARAM_INT);
$sth->execute();
$result = $sth->fetchAll(\PDO::FETCH_ASSOC);
```

up
down
0
luis dot nestesitio at gmail dot com ¶
2 years ago

```
Avoid to use a param with dot like ":table.column".
It will return a error 'PDOException' with message 'SQLSTATE[HY093]: Invalid parameter number:
parameter was not defined' in ...
```

up
down
0
geompse at gmail dot com ¶
6 years ago

```
if you are storing files (or binary data), using PARAM_LOB (and moreover trying to do this with
Oracle), don't miss this page :

http://php.net/manual/en/pdo.lobs.php

You will there notice that PDO-PGSQL and PDO-OCI don't work the same at all : not the same argument
nor the same behaviour.
```

up
down
-8
sergiy dot sokolenko at gmail dot com ¶
6 years ago

```
Note that you cannot mix named and positional parameters in one query:

<?php
$stmt = $conn->prepare('SELECT * FROM employees WHERE name LIKE :name OR email LIKE ?');
$name = 'John%';
$email = 'john%';

$stmt->bindParam(':name', $name);
$stmt->bindParam(1, $email);

$stmt->execute();
```

```
?>

Fatal error: Uncaught exception 'PDOException' with message 'SQLSTATE[HY093]: Invalid parameter
number: mixed named and positional parameters' in ...

Running PHP 5.3.2 on Linux x86-64
```

up
down
-6
ReK_ ¶
6 years ago

```
This confused me for some time because it is never explicitly mentioned, but PDO will automagically
encapsulate parameters for you, so a prepared query that is manually escaped like so:

"INSERT INTO table (column) VALUES (':value');"

Will actually end up being double-quoted and can cause problems.
```

add a note

- PDOStatement
  - bindColumn
  - bindParam
  - bindValue
  - closeCursor
  - columnCount
  - debugDumpParams
  - errorCode
  - errorInfo
  - execute
  - fetch
  - fetchAll
  - fetchColumn
  - fetchObject
  - getAttribute
  - getColumnMeta
  - nextRowset
  - rowCount
  - setAttribute
  - setFetchMode