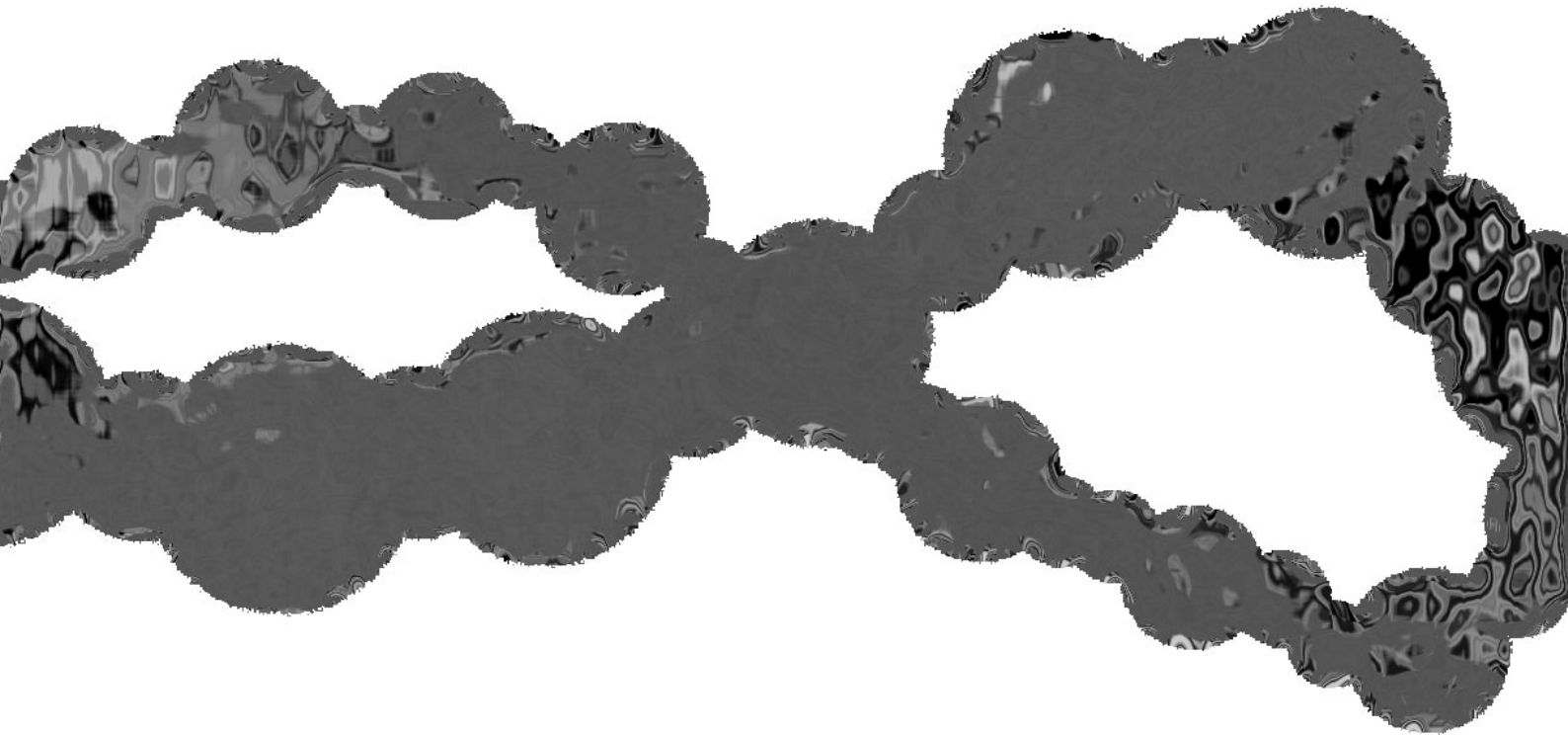


# S PHP imple DB



**Simple mapping database to objects**

# Documentation of use of PHP Simple DB

Introduction and council of use .....	- 3 -
Simple handling of an object.....	- 4 -
Simple handling of a collection.....	- 5 -
Detailed documentation of the objects .....	- 6 -
Method getData_field( \$sName ) .....	- 7 -
Method setData_field( \$sName, \$sValue ) .....	- 8 -
Method getData_object( \$sName ) .....	- 9 -
Method setData_object( \$sName, \$oValue ) .....	- 10 -
Method getData_collection( \$sName ) .....	- 11 -
Method load( \$oData = NULL ) .....	- 12 -
Method save ( \$oData = NULL, \$bForceCreate = false ).....	- 13 -
Method delete( \$bDeleteCollections = false ).....	- 14 -
Method getData_infos() .....	- 15 -
Method getData_values() .....	- 16 -
Method getData_key().....	- 17 -
Detailed documentation of the collections .....	- 18 -
Method setLimit( \$iLimit_page, \$iLength ) .....	- 18 -
Method setOrderBy( \$sOrder, \$bAsc = true ).....	- 19 -
Method count( \$oData ) .....	- 20 -
Method load( \$oData = NULL ) .....	- 21 -
Method save().....	- 22 -
Method delete( \$bDeleteCollections = false ).....	- 23 -
Method addObject( \$oObject ).....	- 24 -
Method delObject( \$oObject, \$bDeleteCollections = false ).....	- 25 -
Method getObjects().....	- 26 -
Method sortByField( \$sName, \$sBy = «asc», \$sType = SORT_NUMERIC ) .....	- 27 -
Method moveOrderField( \$sName, \$iId, \$sBy = «asc» ) .....	- 28 -
Method implodeField( \$sSeparator, \$sField ).....	- 29 -
End .....	- 30 -

## Introduction and council of use

### Introduction :

PHP SIMPLE DB is a tool of assistance to the handling of data in PHP.  
This tool will enable you to generate a set of class representative of your model trade.  
The solution is appeared as a gate Web.

### The Council of use :

We advise you to use in a total way the object of access to the database.

```
<?php
include_once( 'phpsimpledb.class.php' );
Global $oBDD ;
$oBDD = new PHPSIMPLEDB();
?>
```

The data of connection (server, bases, user, password) are in the class “phpsimpledb.class.php”.

If you directly modify the data of connection in the file, those could be lost at the time of the next redeployment.

If you wish to use constants instead of value into clear, to modify your source file (once generated) and to notch the box (not to generate the class “phpsimpledb.class.php”) during your next generation.

## Simple handling of an object

In the following example we show how to update an address e-mail of a user according to his identification number.

```
<?php
// total instantiation
include_once( 'phpsimpledb.class.php' );
Global $oBDD;
$oBDD = new PHPSIMPLEDB();

// creation and loading of "user"
$oUser = $oBDD->getObject( "users" );
$oUser->load( 645 );

// update of "user"
$oUser->setData_field( "email", "admin@phpsimpledb.com" );
$oUser->save();
?>
```

## Simple handling of a collection

In the following example we show how to update a collection of users according to their levels of enablings.

```
<?php
// total instantiation
include_once( 'phpsimplifiedb.class.php' );
Global $oBDD;
$oBDD = new PHPSIMPLEDB();

// creation and loading of the collection of "users"
$oUsers_col = $oBDD->getCollection( "users" );
$oUsers_col->load( array( "level > 80" ) );

// update of the "users" of the collection
foreach( $oUsers_col->getObjects() as $oUser ){
    $oUser->setData_field( "level", 1 );
    $oUser->save();
}
?>
```

## Detailed documentation of the objects

### Necessary meadows:

In order to be able to use the following examples correctly, we recommend to you to take care of the good establishment of the meadows required below.

Note: It is not necessary to set up these necessary meadows.

You must have created a database with the following tables:

Users :

- Id : bigint, auto incrementing
- Name : string(255)
- Level : int

Produits :

- Id : bigint, auto incrementing
- Name : string (255)
- Position : int,
- Users\_id : bigint, foreign key pointing on the table “users”

### Script SQL :

```
CREATE TABLE `test`.`users` (  
  `id` BIGINT NOT NULL AUTO_INCREMENT PRIMARY KEY ,  
  `name` VARCHAR( 255 ) NOT NULL ,  
  `level` INT NOT NULL  
) ;  
  
CREATE TABLE `test`.`produits` (  
  `id` BIGINT NOT NULL AUTO_INCREMENT PRIMARY KEY ,  
  `name` VARCHAR( 255 ) NOT NULL ,  
  `position` INT NULL ,  
  `users_id` BIGINT NOT NULL  
) ;
```

To use the application Web “PHPSIMPLEDB” to create the basic diagram for all the tables, then to bind the table “produced” to the table “users” via the field “users\_id”.

To generate the file and to import it in your project.

If you have difficulties in carry out this file, follow the tutorial “use of the Web application”.

```
// total instantiation  
include_once( 'phpsimpledb.class.php' );  
Global $oBDD;  
$oBDD = new PHPSIMPLEDB();
```

## **Method `getData_field( $sName )`**

Description :

The recovery of an attribute of an object allows.

Parameter :

`$sName`, string : Inform the name of the property of the object

Value of return :

Return a value containing the property requested,

Return "NULL" if the property is non-existent or if it does not have a value

The following example posts the name one "user".

```
<?php
// total instantiation
include_once( 'phpsimpledb.class.php' );
Global $oBDD;
$oBDD = new PHPSIMPLEDB();

// loading of "user"
$oUser = $oBDD->getObject( "users" );
$oUser->load( 21 );

// print the name
print( $oUser->getData_field( "name" ) );

?>
```

## **Method setData\_field( \$sName, \$sValue )**

Description :

This method adds or updates a property of an object

Parameters :

\$sName, string : Represent the name of the field to be updated

\$sValue, string or integer : Represent the value to be updated

Value of return :

Return "true" if the field were updated

Return "false" if the field were not updated

The example following the my up to date name one "user":

```
<?php
// total instantiation
include_once( 'phpsimplifiedb.class.php' );
Global $oBDD;
$oBDD = new PHPSIMPLEDB();

//loading of "user"
$oUser = $oBDD->getObject( "users" );
$oUser->load( 21 ) ;

// update of the name
$oUser->setData_field( "name", "durant" );

// recording of "user"
$oUser->save();
?>
```

## **Method `getData_object( $sName )`**

Description :

This method turns over an object which is implicitly attached to a relative. For example, in a site of real classifieds, a house (product) will be attached to a person (user).

Parameter :

`$sName`, string : Indicate the name of the fields which will identify the attached object

Value of return :

Return an object if this one exists in the object  
Retourne relative "NULL" if the object were not found

The example following return it "user" with a product :

```
<?php
// total instantiation
include_once( 'phpsimpledb.class.php' );
Global $oBDD;
$oBDD = new PHPSIMPLEDB();

//recover a product
$oProduit = $oBDD->getObject( "produits" );
$oProduit->load( 12 );

// recover the object "user" attached to the product
$oUser = $oProduit->getData_object( "users_id" );

?>
```

## **Method setData\_object( \$sName, \$oValue )**

### Description :

This method addition an object "A" with another object "B" which contains a collection of the type implicitly "B". For example, in a site of real classifieds, a house (product) will be attached to a person (user).

### Parameters :

\$sName, string : Indicate the name of the fields which will identify the attached object

\$oValue, object : Indicate the object to be attached

### Value of return :

Return "true" if the addition were carried out successfully

Return "false" if the object could not be added

The example following addition a product to one "user":

```
<?php

//total instantiation
include_once( 'phpsimplifiedb.class.php' );
Global $oBDD;
$oBDD = new PHPSIMPLEDB();

//recovery of "user"
$oUser = $oBDD->getObject( "users" );
$oUser->load( 21 );

//creation of the product
$oProduit = $oBDD->getObject( "produits" );
$oProduit->setData_field( "name", "jolie maison" );

// addition of the product to "using"
$oProduit->setData_object( "users_id", $oUser );

//safeguard product
$oProduit->save();

?>
```

Note: If you try to safeguard new a object bases some, you ensure that all the obligatory values are indicated. If you forget to inform the obligatory fields, your attempt at safeguard will fail.

## **Method `getData_collection( $sName )`**

### Description :

This method turns over a collection belonging to a object.

### Parameter :

`$sName`, string : Indicate the name of the field which will identify the collection.

### Value of return :

Return a collection of object if this one is present in the object (it can be empty, if no child is present).

Return "NULL" if the collection were not found.

The following example posts the name of the products one "user":

```
<?php
//total instantiation
include_once( 'phpsimplifiedb.class.php' );
Global $oBDD;
$oBDD = new PHPSIMPLEDB();

//recovery of "user"
$oUser = $oBDD->getObject( "users" );
$oUser->load( 21 );

// recovery of the collection of the products
$oCol_produits = $oUser->getData_collection( "produits" );

// posting of the names of the products
foreach( $oCol_produits->getObjects() as $oProd ){
    print( "product name : ".$oProd->getData_field( "name" )."<br/>" );
}
?>
```

## **Method load( \$oData = NULL )**

### Description :

This method feeds the properties of a object starting from the database.

### Parameter :

\$oData (optional), (string or integer, array of key / value) : represent either the primary key of the object (string or integer), or array of key / value representing a filter of research or the first occurrence will be returned.

### Value of return :

Return "true" if the loading it passed well.

Return "false" if the filter of research "\$oData" did not make go up any result.

Note: In certain case, where the primary key of a object is not an entirety, certain instabilities will be able to arrive. We recommend to you to prefer the installation of a numerical primary key for each one of your tables.

The following example seeks and posts the name one to use :

```
<?php
//total instanciacion
include_once( 'phpsimpledb.class.php' );
Global $oBDD;
$oBDD = new PHPSIMPLEDB();

//recovery of "user"
$oUser = $oBDD->getObject( "users" );
$oUser->load( 12 ) ;

//print name of "user"
print( $oUser->getData_field( "name" ) );
?>
```

## **Method save ( \$aData = NULL, \$bForceCreate = false )**

### Description :

This method safeguards the properties of a object in the database.

### Parameters :

**\$aData** (optional), array of key /value : This parameter can contain key pairs/value or key/object. The properties of the object will be updated before its recording.

**\$bForceCreate** (optional), Boolean : This parameter is used during the creation of object not comprising a numerical primary key. By putting the parameter at "true", the object will be created in the database.

### Value of return :

No parameter of return.

The following example illustrates the recording of a product with an update of certain properties :

```
<?php
//total instantiation
include_once( 'phpsimpledb.class.php' );
Global $oBDD;
$oBDD = new PHPSIMPLEDB();

//get a product
$oProduit = $oBDD->getObject( "produits" );
$oProduit->load( 2 );

// creation new "user"
$oUser = $oBDD->getObject( "users" );
$oUser->setData_field( "name", "John Do" );
$oUser->setData_field( "level", 80 );

// update of the product
$oProduit->save( array( "name"=>" Pretty house ", "users_id"=>$oUser ) );
?>
```

## **Method delete( \$bDeleteCollections = false )**

### Description :

This method removes a object of the database.

### Parameter :

`$bDeleteCollection` (optional), Boolean: This parameter allows the release of the suppression of the objects in cascade, all the collections of the objects children will be they also removed (attention, the concept of level of suppression does not exist).

### Value of return :

Return "true" if the object could be removed.

Return "false" if the object could not be removed (if this one is not present in the base of data, "false" is returned).

The following example illustrates the removal of a product :

```
<?php
//total instantiation
include_once( 'phpsimpledb.class.php' );
Global $oBDD;
$oBDD = new PHPSIMPLEDB();

// recovery of the product
$oProduit = $oBDD->getObject( "produits" ) ;
$oProduit->load( 12 ) ;

// remove the product
$oProduit->delete() ;

?>
```

## **Method `getData_infos()`**

Description :

This method turns over a key table/value containing the related informations to the table. This table contains the name of the table, the name of the primary key and the description of the table.

Value of return :

Return array of key / value containing information of the table.

The following example posts the name and the primary key of the table of the products :

```
<?php
//total instantiation
include_once( 'phpsimplifiedb.class.php' );
Global $oBDD;
$oBDD = new PHPSIMPLEDB();

// recovery of a "product" object
$oProduit = $oBDD->getObject( "produits" );

// recovery of information
$oData_table = $oProduit->getData_infos();

//print a informations
print( "Table : ".$oData_table[ "table" ]."<br/>" );
print( "Key : ".$oData_table[ "primarykey" ]."<br/>" );
?>
```

## **Method `getData_values()`**

Description :

This method turns over of array key / value containing all the values of the object indexed according to the name of the field.

Value of return :

Return of array key / value containing the name of the fields in index and their associated values.

The following example posts the whole of the properties of a produced object :

```
<?php

//total instantiation
include_once( 'phpsimpledb.class.php' );
Global $oBDD;
$oBDD = new PHPSIMPLEDB();

// recovery of the product
$oProduit = $oBDD->getObject( "produits" );
$oProduit->load( 13 );

// recovery of all the values of the product
$oValues = $oProduit->getData_values();

// posting of the values
foreach( $oValues as $sKey=>$sValue ){
    print( $sKey." : ".$sValue."<br/>" );
}

?>
```

## **Method `getData_key()`**

Description :

This method returns the primary key of the object. If this one is non-existent, "NULL" will be returned.

Value of return :

Return the value of the primary key of the object.

Return "NULL" if the value of the primary key of the object is null.

The following example posts the primary key of a product :

```
<?php
//total instantiation
include_once( 'phpsimplifiedb.class.php' );
Global $oBDD;
$oBDD = new PHPSIMPLEDB();

// recovery of the product
$oProduit = $oBDD->getObject( "produits" );
$oProduit->load( 13 );

// print the primary key
print( "Key : ".$oProduit->getData_key() );
?>
```

## Detailed documentation of the collections

### ***Method setLimit( \$iLimit\_page, \$iLength )***

#### Description :

The limitation amongst objects allows turned over by the collection.

#### Parameters :

`$iLimit_page`, integer : This parameter awaits a number of page (minimum 1)

`$iLength`, integer : This parameter determines the number of beach to return

#### Value of return :

No parameter of return.

The following example recovers the 10 first “users” according to their levels of authorization :

```
<?php
//total instantiation
include_once( 'phpsimpledb.class.php' );
Global $oBDD;
$oBDD = new PHPSIMPLEDB();

// loading of the 10 first “users” according to “level = 1”
$oUsers_col = $oBDD->getCollection( "users" );
$oUsers_col->setLimit( 1, 10 );
$oUsers_col->load( array( "level = 1" ) );
?>
```

## **Method `setOrderBy( $sOrder, $bAsc = true )`**

Description :

This Method parameter an order of classification of the lines of a collection. The classification can be done according to a field, in the ascending order or descendant.

Parameters :

`$sOrder`, string : Determine the field in which the objects of the collection must be classified.

`$bAsc` (optional), Boolean: Determine the direction of sorting of the collection, "true" for ascending. By defect the order of classification will be ascending if the second parameter and omitted.

Value of return :

No parameter of return.

The following example recovers a list of "user" alphabetically classified at once "name":

```
<?php
//total instantiation
include_once( 'phpsimpledb.class.php' );
Global $oBDD;
$oBDD = new PHPSIMPLEDB();

// loading of the "users" alphabetically
$oUsers_col = $oBDD->getCollection( "users" );
$oUsers_col->setOrderBy( "name" );
$oUsers_col->load();
?>
```

## **Method count( \$oData )**

Description :

Allows to count the number of element of an expression.

Parameter :

\$oData, (string or array of string) : The parameter will be either of standard table of value, or of standard string. If this one is of standard table, the elements of the table will be assembled with operator SQL "AND".

Value of return :

Return a integer representing the result of the counting of the elements.

The following example counts the number of "users" having a level of authorization equalizes to 1 :

```
<?php
//total instantiation
include_once( 'phpsimplifiedb.class.php' );
Global $oBDD;
$oBDD = new PHPSIMPLEDB();

// count the number of "users" nonvalid
$oUsers_col = $oBDD->getCollection( "users" );
$iNbr_users = $oUsers_col->count( "level = 1" );

?>
```

## **Method load( \$oData = NULL )**

Description :

The loading of a collection allows.

Parameter :

\$oData (optional), (string or array of string) : The parameter will be either of standard table of value, or of standard String. If this one is of standard table, the elements of the table will be assembled with operator SQL "AND". If this parameter is omitted, all the recordings of the collection will be returned.

Value of return :

Return "true" if the loading of the collection is carried out correctly.

Return "false" if the collection did not go up any recording.

The following example recovers the "users" pertaining to level 1 and not bearing the name of "during" :

```
<?php
//total instantiation
include_once( 'phpsimpledb.class.php' );
Global $oBDD;
$oBDD = new PHPSIMPLEDB();

// loading of the "users" with "name != 'durant'" and "level = 1"
$oUsers_col = $oBDD->getCollection( "users" );
$oUsers_col->load( array( "name != 'durant'", "level = 1" ) );

?>
```

## **Method save()**

Description :

The recording of the objects of a collection allows.

Value of return :

No parameter of return.

The following example modifies the level of authorization of the “users” pertaining to level 1 :

```
<?php

//total instantiation
include_once( 'phpsimpledb.class.php' );
Global $oBDD;
$oBDD = new PHPSIMPLEDB();

// recover all the "users" with "level = 1"
$oUsers_col = $oBDD->getCollection( "users" );
$oUsers_col->load( "level = 1" );

// modification of activation
Foreach( $oUsers_col->getObjects() as $oUser ){
    $oUser->setData_field( "level", 2 );
}

// recording of the modifications
$oUsers_col->save();

?>
```

## **Method delete( \$bDeleteCollections = false )**

Description :

The suppression of the objects of a collection allows.

Parameter :

\$bDeleteCollection (optional), Boolean: This parameter allows the release of the suppression of the objects in cascade, all the collections of the objects children will be they also removed (attention, the concept of level of suppression does not exist).

Value of return :

No parameter of return.

The following example removes the list of the “users” having a level of authorization equalizes to 0 :

```
<?php
//total instantiation
include_once( 'phpsimplifiedb.class.php' );
Global $oBDD;
$oBDD = new PHPSIMPLEDB();

// recover all the "users" with "level = 0"
$oUsers_col = $oBDD->getCollection( "users" );
$oUsers_col->load( "level = 0" );

// remove the "users"
$oUsers_col->delete();

?>
```

## **Method addObject( \$Object )**

Description :

The addition of a object allows a collection.

If the collection belongs to a object (as "product" belongs to "users"), the added object will be automatically connected to the collection.

Parameter :

\$Object, object : This parameter represents the object to be added

Value of return :

Return "true" if the addition with the collection occurred well

Return "false" if the object passed in parameter does not correspond to the type of the collection, or if the object does not contain an identifier.

The example following addition a product to the collection of the products of "users" :

```
<?php
//total instantiation
include_once( 'phpsimplifiedb.class.php' );
Global $oBDD;
$oBDD = new PHPSIMPLEDB();

// recover all the "products" one "to use"
$oUser = $oBDD->getObject( "users" );
$oUser->load( 2 ) ;

// recover the collection of the products of "user"
$oProduits_col = $oUser->getData_collection( "produits" );

// creation of a new product
$oProduit = $oBDD->getObject( "produits" );
$oProduit->setData_field( "name", "Pavillon" );

// addition of the product to the collection of the products of "using"
$oProduits_col->addObject( $oProduit ) ;
$oProduits_col->save();
?>
```

## **Method delObject( \$Object, \$bDeleteCollections = false )**

Description :

Allows to remove a object of a collection.

Parameters :

\$Object, object : This parameter represents the object to be removed.

\$bDeleteCollection (optional), Boolean: This parameter allows the release of the suppression of the objects in cascade, all the collections of the objects children will be they also removed (attention, the concept of level of suppression does not exist).

Value of return :

No parameter of return.

The following example removes a user of a collection of "users" :

```
<?php
//total instantiation
include_once( 'phpsimpledb.class.php' );
Global $oBDD;
$oBDD = new PHPSIMPLEDB();

// recover a collection of "users"
$oUsers_col = $oBDD->getCollection( "users" );
$oUsers_col->load( "level = 2" );

// recover the first recording
$oUsers = $oUsers_col->getObjects();
$oUser = $oUsers[ 0 ];

// remove it "user"
$oUsers_col->delObject( $oUser );

?>
```

## **Method `getObjects()`**

Description :

This Method turns over a table containing a list of objects.

Value of return :

This Method turns over a table containing unRetourne a table satisfying a list with objects.

If no element is found, the returned table will be empty.

The following example recovers a list of "users" :

```
<?php
//total instantiation
include_once( 'phpsimpledb.class.php' );
Global $oBDD;
$oBDD = new PHPSIMPLEDB();

// recover a collection of "users"
$oUsers_col = $oBDD->getCollection( "users" );
$oUsers_col->load();

// recover the list of the "users"
$oUsers = $oUsers_col->getObjects();
?>
```

## **Method `sortByField( $sName, $sBy = «asc», $sType = SORT_NUMERIC )`**

Description :

Allows to sort a collection on a field.

Parameters :

`$sName`, string : e of the field on which the sorting will be made

`$sBy` (optional), string : Indicate the direction of the sorting,

«asc», indicating an ascending sorting

«desc», indicating a tri descendant

By defect this parameter is equal to «asc»

`$sType` (optional), constant : Inform the type of type to be carried out

`SORT_NUMERIC`, indicating a kind of numerical field

`SORT_STRING`, indicating a kind of field string

Value of return :

No parameter of return.

The following example posts a list of “users” sorted by name in a downward way :

```
<?php
//total instantiation
include_once( 'phpsimplifiedb.class.php' );
Global $oBDD;
$oBDD = new PHPSIMPLEDB();

// recover a collection of "users"
$oUsers_col = $oBDD->getCollection( "users" );
$oUsers_col->load();

// Tri descendant at once "name"
$oUsers_col->sortByField( "name", "desc", SORT_STRING );

//print a "users"
foreach($oUsers_col->getObjects() as $oUser ){
    print( "name : ".$oUser->getData_field( "name" )."<br/>" );
}
?>
```

## **Method `moveOrderField( $sName, $sId, $sBy = «asc» )`**

### Description :

Allows to assemble or descend a value from a field of the integer type in a object by respecting the order of the list.

### Parameters :

`$sName`, string : Indicate the name of the field which will be used for displacement

`$sId`, integer : Indicate the identifier of the object

`$sBy` (optional), string : Indicate the direction of displacement

«asc», indicating an ascending sorting

«desc», indicating a tri descendant

By defect this parameter is equal to «asc»

### Value of return :

Return "true" if displacement it passed well.

Return "false" if the collection does not contain a object or if the object were not found.

The following example changes the position to the bottom of a product of a bundle of commodities one "user" :

```
<?php
//total instantiation
include_once( 'phpsimpledb.class.php' );
Global $oBDD;
$oBDD = new PHPSIMPLEDB();

//loading of "user"
$oUser = $oBDD->getObject( "users" );
$oUser->load( 12 );

// loading of the products of the "users" for construction of the
collection
$oProduits_col = $oUser->getData_collection( "produits" );

// displacement of the position of the product
$oProduits_col->moveOrderField( "position", 4, "desc" );

// print a "products" with their position
foreach( $oProduits_col->getObjects() as $oProduit ){
    print( "id : ".$oProduit->getData_field( "id" ).", nom : ".$oProduit-
>getData_field( "name" ).", position : ".$oProduit-
>getData_field( "position" )."<br/>" );
}

// store the collection
$oProduits_col->save();

?>
```

## **Method implodeField( \$sSeparator, \$sField )**

Description :

Return a string containing a concatenation of the desired field.

Parameters :

\$sSeparator, string : Indicate the separator of concatenation

\$sField, string : Indicate the name of the field to be recovered

Value of return :

Return a string containing a concatenation of the values of a field.

The following example recovers poster the names of the “users” according to their levels of authorization, with a separation with points commas “;”:

```
<?php
//total instantiation
include_once( 'phpsimpledb.class.php' );
Global $oBDD;
$oBDD = new PHPSIMPLEDB();

// recover a collection of "users" not validated
$oUsers_col = $oBDD->getCollection( "users" );
$oUsers_col->load( "level = 2" );

// posting of the name separated by points commas ";"
print( $oUsers_col->implodeField( ";", "name" ) );

?>
```

**End**

If this documentation seems to you incomplete or if you have of evil to understand certain points, to address your remarks to : **[contact@phpsimpledb.com](mailto:contact@phpsimpledb.com)**