Class/Object Information

Introduction

These functions allow you to obtain information about classes and instance objects. You can obtain the name of the class to which an object belongs, as well as its member properties and methods. Using these functions, you can find out not only the class membership of an object, but also its parentage (i.e. what class is the object class extending).

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

This extension has no configuration directives defined in *php.ini*.

Resource Types

This extension has no resource types defined.

Predefined Constants

This extension has no constants defined.

Examples

In this example, we first define a base class and an extension of the class. The base class describes a general vegetable, whether it is edible or not and what is its color. The subclass *Spinach* adds a method to cook it and another to find out if it is cooked.

Example #1 - classes.inc

```
<?php

// base class with member properties and methods
class Vegetable {

  var $edible;
  var $color;

  function Vegetable($edible, $color="green")
  {
      $this->edible = $edible;
      $this->color = $color;
  }

  function is_edible()
  {
      return $this->edible;
  }

  function what_color()
  {
      return $this->color;
  }

} // end of class Vegetable

// extends the base class
class Spinach extends Vegetable {

  var $cooked = false;

  function Spinach()
  {
      $this->Vegetable(true, "green");
  }

  function cook_it()
  {
      $this->cooked = true;
  }

  function is_cooked()
  {
      return $this->cooked;
  }
```

```
} // end of class Spinach

?>
```

We then instantiate 2 objects from these classes and print out information about them, including their class parentage. We also define some utility functions, mainly to have a nice printout of the variables.

Example #2 - test_script.php

```
<pre>
<?php

include "classes.inc";

// utility functions

function print_vars($obj)
{
foreach (get_object_vars($obj) as $prop => $val) {
   echo "\t$prop = $val\n";
}
}

function print_methods($obj)
{
$arr = get_class_methods(get_class($obj));
foreach ($arr as $method) {
   echo "\tfunction $method()\n";
}
}

function class_parentage($obj, $class)
{
if (is_subclass_of($GLOBALS[$obj], $class)) {
   echo "Object $obj belongs to class " . get_class($$obj);
   echo " a subclass of $class\n";
} else {
   echo "Object $obj does not belong to a subclass of $class\n";
}
}

// instantiate 2 objects

$veggie = new Vegetable(true, "blue");
$leafy = new Spinach();

// print out information about objects
echo "veggie: CLASS " . get_class($veggie) . "\n";
echo "leafy: CLASS " . get_class($leafy);
echo ", PARENT " . get_parent_class($leafy) . "\n";

// show veggie properties
echo "\nveggie: Properties\n";
print_vars($veggie);
```

```
// and leafy methods
echo "\nleafy: Methods\n";
print_methods($leafy);

echo "\nParentage:\n";
class_parentage("leafy", "Spinach");
class_parentage("leafy", "Vegetable");
?>
</pre>
```

One important thing to note in the example above is that the object *$leafy* is an instance of the class Spinach which is a subclass of Vegetable, therefore the last part of the script above will output:

```
[...]
Parentage:
Object leafy does not belong to a subclass of Spinach
Object leafy belongs to class spinach a subclass of Vegetable
```

Classes/Object Functions

call_user_method_array

call_user_method_array -- Call a user method given with an array of parameters [deprecated]

Description

mixed **call_user_method_array** (string `$method_name`, object `&$obj`, array `$params`)

Warning
The call_user_method_array() function is deprecated as of PHP 4.1.0.

Examples

Example #3 - call_user_method_array() alternative

```
<?php
call_user_func_array(array($obj, $method_name), $params);
call_user_func_array(array(&$obj, $method_name), $params); // PHP 4
?>
```

See Also

- call_user_func_array()
- call_user_func()

call_user_method

call_user_method -- Call a user method on an specific object [deprecated]

Description

mixed **call_user_method** (string `$method_name`, object `&$obj` [, mixed `$parameter` [, mixed `$...`]])

Warning
The call_user_method() function is deprecated as of PHP 4.1.0.

Examples

Example #4 - call_user_method() alternative

```
<?php
call_user_func(array($obj, $method_name), $parameter /* , ... */);
call_user_func(array(&$obj, $method_name), $parameter /* , ... */); // PHP 4
?>
```

See Also

- call_user_func_array()
- call_user_func()

class_exists

class_exists -- Checks if the class has been defined

Description

bool **class_exists** (string `$class_name` [, bool `$autoload`])

This function checks whether or not the given class has been defined.

Parameters

`class_name`
The class name. The name is matched in a case-insensitive manner.

`autoload`
Whether or not to call __autoload by default. Defaults to **TRUE**.

Return Values

Returns **TRUE** if *`class_name`* is a defined class, **FALSE** otherwise.

ChangeLog

Version	Description
5.0.2	No longer returns **TRUE** for defined interfaces. Use interface_exists().
5.0.0	The *`autoload`* parameter was added.

Examples

Example #5 - class_exists() example

```
<?php
// Check that the class exists before trying to use it
if (class_exists('MyClass')) {
   $myclass = new MyClass();
}
```

```
?>
```

Example #6 - *autoload* parameter example

```
<?php
function __autoload($class)
{
   include($class . '.php');

   // Check to see whether the include declared the class
   if (!class_exists($class, false)) {
       trigger_error("Unable to load class: $class", E_USER_WARNING);
   }
}

if (class_exists('MyClass')) {
   $myclass = new MyClass();
}

?>
```

See Also

- function_exists()
- interface_exists()
- get_declared_classes()

get_class_methods

get_class_methods -- Gets the class methods' names

Description

array **get_class_methods** (mixed `$class_name`)

Gets the class methods names.

Parameters

`class_name`
The class name or an object instance

Return Values

Returns an array of method names defined for the class specified by *`class_name`*. In case of an error, it returns **NULL**.

ChangeLog

Version	Description
5.0.0	As of PHP 5, this function returns the name of the methods as they were declared (case-sensitive). In PHP 4 they were lowercased.
4.0.6	The ability of specifying the object itself has been added.

Examples

Example #7 - get_class_methods() example

```
<?php

class myclass {
   // constructor
   function myclass()
```

```
    {
        return(true);
    }

    // method 1
    function myfunc1()
    {
        return(true);
    }

    // method 2
    function myfunc2()
    {
        return(true);
    }
}

$class_methods = get_class_methods('myclass');
// or
$class_methods = get_class_methods(new myclass());

foreach ($class_methods as $method_name) {
    echo "$method_name\n";
}

?>
```

The above example will output:

```
myclass
myfunc1
myfunc2
```

See Also

- get_class()
- get_class_vars()
- get_object_vars()

get_class_vars

get_class_vars -- Get the default properties of the class

Description

array **get_class_vars** (string `$class_name`)

Get the default properties of the given class.

Parameters

`class_name`
The class name

Return Values

Returns an associative array of default public properties of the class. The resulting array elements are in the form of *varname => value*.

ChangeLog

Version	Description
Prior to 4.2.0	Uninitialized class variables will not be reported by get_class_vars()

Examples

Example #8 - get_class_vars() example

```
<?php

class myclass {

   var $var1; // this has no default value...
   var $var2 = "xyz";
   var $var3 = 100;
   private $var4; // PHP 5

   // constructor
   function myclass() {
```

```
        // change some properties
        $this->var1 = "foo";
        $this->var2 = "bar";
        return true;
    }

}

$my_class = new myclass();

$class_vars = get_class_vars(get_class($my_class));

foreach ($class_vars as $name => $value) {
    echo "$name : $value\n";
}

?>
```

The above example will output:

```
// Before PHP 4.2.0
var2 : xyz
var3 : 100

// As of PHP 4.2.0
var1 :
var2 : xyz
var3 : 100
```

See Also

- get_class_methods()
- get_object_vars()

get_class

get_class -- Returns the name of the class of an object

Description

string **get_class** ([object `$object`])

Gets the name of the class of the given *`object`*.

Parameters

`object`
The tested object

Return Values

Returns the name of the class of which *`object`* is an instance. Returns **FALSE** if *`object`* is not an object.

ChangeLog

Version	Description
Since 5.0.0	The class name is returned in it's original notation.
Since 5.0.0	The *`object`* parameter is optional if called from the object's method.

Examples

Example #9 - Using get_class()

```
<?php

class foo {
   function name()
   {
       echo "My name is " , get_class($this) , "\n";
   }
```

```
}

// create an object
$bar = new foo();

// external call
echo "Its name is " , get_class($bar) , "\n";

// internal call
$bar->name();

?>
```

The above example will output:

```
Its name is foo
My name is foo
```

Example #10 - Using get_class() in superclass

```
<?php

abstract class bar {
 public function __construct()
 {
  var_dump(get_class($this));
  var_dump(get_class());
 }
}

class foo extends bar {
}

new foo;

?>
```

The above example will output:

```
string(3) "foo"
string(3) "bar"
```

See Also

- get_parent_class()
- gettype()
- is_subclass_of()

get_declared_classes

get_declared_classes -- Returns an array with the name of the defined classes

Description

array **get_declared_classes** (void)

Gets the declared classes.

Return Values

Returns an array of the names of the declared classes in the current script.

Note

In PHP 4.0.1, three extra classes are returned at the beginning of the array: stdClass (defined in *Zend/zend.c*), OverloadedTestClass (defined in *ext/standard/basic_functions.c*) and Directory (defined in *ext/standard/dir.c*).

Also note that depending on what extensions you have compiled or loaded into PHP, additional classes could be present. This means that you will not be able to define your own classes using these names. There is a list of predefined classes in the Predefined Classes section of the appendices.

Examples

Example #11 - get_declared_classes() example

```
<?php
print_r(get_declared_classes());
?>
```

The above example will output something similar to:

```
Array
(
    [0] => stdClass
    [1] => __PHP_Incomplete_Class
    [2] => Directory
)
```

See Also

- class_exists()
- get_declared_interfaces()
- get_defined_functions()

get_declared_interfaces

get_declared_interfaces -- Returns an array of all declared interfaces

Description

array **get_declared_interfaces** (void)

Gets the declared interfaces.

Return Values

Returns an array of the names of the declared interfaces in the current script.

Examples

Example #12 - get_declared_interfaces() **example**

```
<?php
print_r(get_declared_interfaces());
?>
```

The above example will output something similar to:

```
Array
(
    [0] => Traversable
    [1] => IteratorAggregate
    [2] => Iterator
    [3] => ArrayAccess
    [4] => reflector
    [5] => RecursiveIterator
    [6] => SeekableIterator
)
```

See Also

- get_declared_classes()
- class_implements()

get_object_vars

get_object_vars -- Gets the public properties of the given object

Description

array **get_object_vars** (object `$object`)

Gets the public properties of the given *`object`*.

Parameters

`object`
An object instance.

Return Values

Returns an associative array of defined object public properties for the specified *`object`*. If a property have not been assigned a value, it will be returned with a **NULL** value.

ChangeLog

Version	Description
prior to 4.2.0	If the variables declared in the class of which the *`object`* is an instance, have not been assigned a value, those will not be returned in the array

Examples

Example #13 - Use of get_object_vars()

```
<?php

class foo {
   private $a;
   public $b = 1;
   public $c;
   private $d;
   static $e;
```

```
}

$test = new foo;
var_dump(get_object_vars($test));

?>
```

The above example will output:

```
array(2) {
 ["b"]=>
 int(1)
 ["c"]=>
 NULL
}
```

See Also

- get_class_methods()
- get_class_vars()

get_parent_class

get_parent_class -- Retrieves the parent class name for object or class

Description

string **get_parent_class** ([mixed `$object`])

Retrieves the parent class name for object or class.

Parameters

`object`
The tested object or class name

Return Values

Returns the name of the parent class of the class of which *`object`* is an instance or the name.

Note
If the object does not have a parent **FALSE** will be returned.

If called without parameter outside object, this function returns **FALSE**.

ChangeLog

Version	**Description**
Before 5.1.0	If called without parameter outside object, this function would have returned **NULL** with a warning.
Since 5.0.0	The *`object`* parameter is optional if called from the object's method.
Since 4.0.5	If *`object`* is a string, returns the name of the parent class of the class with that name.

Examples

Example #14 - Using get_parent_class()

```
<?php

class dad {
   function dad()
   {
   // implements some logic
   }
}

class child extends dad {
   function child()
   {
       echo "I'm " , get_parent_class($this) , "'s son\n";
   }
}

class child2 extends dad {
   function child2()
   {
       echo "I'm " , get_parent_class('child2') , "'s son too\n";
   }
}

$foo = new child();
$bar = new child2();

?>
```

The above example will output:

```
I'm dad's son
I'm dad's son too
```

See Also

- get_class()
- is_subclass_of()

interface_exists

interface_exists -- Checks if the interface has been defined

Description

bool **interface_exists** (string `$interface_name` [, bool `$autoload`])

Checks if the given interface has been defined.

Parameters

`interface_name`
The interface name

`autoload`
Whether to call __autoload or not by default

Return Values

Returns **TRUE** if the interface given by *`interface_name`* has been defined, **FALSE** otherwise.

Examples

Example #15 - interface_exists() example

```
<?php
// Check the interface exists before trying to use it
if (interface_exists('MyInterface')) {
   class MyClass implements MyInterface
   {
       // Methods
   }
}

?>
```

See Also

- class_exists()

is_a

is_a -- Checks if the object is of this class or has this class as one of its parents

Description

bool **is_a** (object `$object`, string `$class_name`)

Checks if the given *`object`* is of this class or has this class as one of its parents.

Note
The is_a() function is deprecated as of PHP 5 in favor of the instanceof type operator.

Parameters

`object`
: The tested object

`class_name`
: The class name

Return Values

Returns **TRUE** if the object is of this class or has this class as one of its parents, **FALSE** otherwise.

Examples

Example #16 - is_a() example

```
<?php
// define a class
class WidgetFactory
{
 var $oink = 'moo';
}

// create a new object
$WF = new WidgetFactory();

if (is_a($WF, 'WidgetFactory')) {
 echo "yes, \$WF is still a WidgetFactory\n";
}
```

```
?>
```

Example #17 - Using the *instanceof* operator in PHP 5

```
<?php
if ($WF instanceof WidgetFactory) {
   echo 'Yes, $WF is a WidgetFactory';
}
?>
```

See Also

- get_class()
- get_parent_class()
- is_subclass_of()

is_subclass_of

is_subclass_of -- Checks if the object has this class as one of its parents

Description

bool **is_subclass_of** (mixed `$object`, string `$class_name`)

Checks if the given *`object`* has the class *`class_name`* as one of its parents.

Parameters

`object`
A class name or an object instance

`class_name`
The class name

Return Values

This function returns **TRUE** if the object *`object`*, belongs to a class which is a subclass of *`class_name`*, **FALSE** otherwise.

ChangeLog

Version	Description
5.0.3	You may also specify the *`object`* parameter as a string (the name of the class)

Examples

Example #18 - is_subclass_of() example

```
<?php
// define a class
class WidgetFactory
{
 var $oink = 'moo';
}
```

```
// define a child class
class WidgetFactory_Child extends WidgetFactory
{
 var $oink = 'oink';
}

// create a new object
$WF = new WidgetFactory();
$WFC = new WidgetFactory_Child();

if (is_subclass_of($WFC, 'WidgetFactory')) {
 echo "yes, \$WFC is a subclass of WidgetFactory\n";
} else {
 echo "no, \$WFC is not a subclass of WidgetFactory\n";
}


if (is_subclass_of($WF, 'WidgetFactory')) {
 echo "yes, \$WF is a subclass of WidgetFactory\n";
} else {
 echo "no, \$WF is not a subclass of WidgetFactory\n";
}


// usable only since PHP 5.0.3
if (is_subclass_of('WidgetFactory_Child', 'WidgetFactory')) {
 echo "yes, WidgetFactory_Child is a subclass of WidgetFactory\n";
} else {
 echo "no, WidgetFactory_Child is not a subclass of WidgetFactory\n";
}
?>
```

The above example will output:

```
yes, $WFC is a subclass of WidgetFactory
no, $WF is not a subclass of WidgetFactory
yes, WidgetFactory_Child is a subclass of WidgetFactory
```

See Also

- get_class()
- get_parent_class()
- is_a()

method_exists

method_exists -- Checks if the class method exists

Description

bool **method_exists** (object `$object`, string `$method_name`)

Checks if the class method exists in the given *`object`*.

Parameters

`object`
An object instance

`method_name`
The method name

Return Values

Returns **TRUE** if the method given by *`method_name`* has been defined for the given *`object`*, **FALSE** otherwise.

Examples

Example #19 - method_exists() example

```
<?php
$directory = new Directory('.');
var_dump(method_exists($directory,'read'));
?>
```

The above example will output:

```
bool(true)
```

Example #20 - Static method_exists() example

```
<?php
$directory = new Directory('.');
var_dump(method_exists('Directory','read'));
?>
```

The above example will output:

```
bool(true)
```

See Also

- function_exists()
- is_callable()

property_exists

property_exists -- Checks if the object or class has a property

Description

bool **property_exists** (mixed `$class`, string `$property`)

This function checks if the given *`property`* exists in the specified class (and if it is accessible from the current scope).

Note
As opposed with isset(), property_exists() returns **TRUE** even if the property has the value **NULL**.

Parameters

`class`
The class name or an object of the class to test for

`property`
The name of the property

Return Values

Returns **TRUE** if the property exists, **FALSE** if it doesn't exist or **NULL** in case of an error.

Examples

Example #21 - A property_exists() example

```
<?php

class myClass {
   public $mine;
   private $xpto;

   static function test() {
       var_dump(property_exists('myClass', 'xpto')); // true, it can be
accessed from here
   }
}
```

```
var_dump(property_exists('myClass', 'mine'));   //true
var_dump(property_exists(new myClass, 'mine')); //true
var_dump(property_exists('myClass', 'xpto'));   //false, isn't public
myClass::test();

?>
```

See Also

- method_exists()