Standard PHP Library (SPL)

Introduction

SPL is a collection of interfaces and classes that are meant to solve standard problems.

Tip
A more detailed documentation of SPL can be found » here.

Installing/Configuring

Requirements

No external libraries are needed to build this extension.

Installation

This extension is available and compiled by default in PHP 5.0.0.

Note
As of PHP 5.3.0 this extension can no longer be disabled and is therefore always available.

Runtime Configuration

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

Resource Types

This extension has no resource types defined.

Predefined Constants

The constants below are defined by this extension, and will only be available when the extension has either been compiled into PHP or dynamically loaded at runtime.

Warning
SPL uses class constants since PHP 5.1. Prior releases use global constants in the form **RIT_LEAVES_ONLY**.

RecursiveIteratorIterator::LEAVES_ONLY (integer)

RecursiveIteratorIterator::SELF_FIRST (integer)

RecursiveIteratorIterator::CHILD_FIRST (integer)

CachingIterator::CALL_TOSTRING (integer)

CachingIterator::CATCH_GET_CHILD (integer)

Datastructures

SPL provies a set of standard datastructures. They are grouped here by their underlying implementation which usually defines their general field of application.

Doubly Linked Lists

A Doubly Linked List (DLL) is a list of nodes linked in both directions to each others. Iterator's operations, access to both ends, addition or removal of nodes have a cost of O(1) when the underlying structure is a DLL. It hence provides a decent implementation for stacks and queues.

- SplDoublyLinkedList
  - SplStack
  - SplQueue

Heaps

Heaps are tree-like structures that follow the heap-property: each node is greater than or equal to its children, when compared using the implemented compare method which is global to the heap.

- SplHeap
  - SplMaxHeap
  - SplMinHeap

- SplPriorityQueue

SPL Functions

class_implements

class_implements -- Return the interfaces which are implemented by the given class

Description

array **class_implements** (mixed `$class` [, bool `$autoload`])

This function returns an array with the names of the interfaces that the given *`class`* and its parents implement.

Parameters

`class`
An object (class instance) or a string (class name).

`autoload`
Whether to allow this function to load the class automatically through the __autoload magic method. Defaults to **TRUE**.

Return Values

An array on success, or **FALSE** on error.

ChangeLog

Version	Description
5.1.0	Added the option to pass the *`class`* parameter as a string. Added the *`autoload`* parameter.

Examples

Example #1 - class_implements() example

```
<?php

interface foo { }
class bar implements foo {}
```

```
print_r(class_implements(new bar));

// since PHP 5.1.0 you may also specify the parameter as a string
print_r(class_implements('bar'));


function __autoload($class_name) {
  require_once $class_name . '.php';
}

// use __autoload to load the 'not_loaded' class
print_r(class_implements('not_loaded', true));

?>
```

The above example will output something similar to:

```
Array
(
   [foo] => foo
)

Array
(
   [interface_of_not_loaded] => interface_of_not_loaded
)
```

See Also

- class_parents()
- get_declared_interfaces()

class_parents

class_parents -- Return the parent classes of the given class

Description

array **class_parents** (mixed `$class` [, bool `$autoload`])

This function returns an array with the name of the parent classes of the given *`class`*.

Parameters

`class`
An object (class instance) or a string (class name).

`autoload`
Whether to allow this function to load the class automatically through the __autoload magic method. Defaults to **TRUE**.

Return Values

An array on success, or **FALSE** on error.

ChangeLog

Version	Description
5.1.0	Added the option to pass the *`class`* parameter as a string. Added the *`autoload`* parameter.

Examples

Example #2 - class_parents() example

```
<?php

class foo { }
class bar extends foo {}

print_r(class_parents(new bar));
```

```
// since PHP 5.1.0 you may also specify the parameter as a string
print_r(class_parents('bar'));


function __autoload($class_name) {
  require_once $class_name . '.php';
}

// use __autoload to load the 'not_loaded' class
print_r(class_parents('not_loaded', true));
?>
```

The above example will output something similar to:

```
Array
(
   [foo] => foo
)

Array
(
   [parent_of_not_loaded] => parent_of_not_loaded
)
```

See Also

- class_implements()

iterator_count

iterator_count -- Count the elements in an iterator

Description

int **iterator_count** (IteratorAggregate `$iterator`)

Warning
This function is currently not documented; only its argument list is available.

Count the elements in an iterator.

Parameters

`iterator`
The iterator being counted.

Return Values

The number of elements in *`iterator`*.

iterator_to_array

iterator_to_array -- Copy the iterator into an array

Description

array **iterator_to_array** (IteratorAggregate `$iterator` [, bool `$use_keys`])

Warning
This function is currently not documented; only its argument list is available.

Count the elements in an iterator.

Parameters

`iterator`
The iterator being counted.

`use_keys`

Return Values

The number of elements in *`iterator`*.

spl_autoload_call

spl_autoload_call -- Try all registered __autoload() function to load the requested class

Description

void **spl_autoload_call** (string `$class_name`)

This function can be used to manually search for a class or interface using the registered __autoload functions.

Parameters

`class_name`
: The class name being searched.

Return Values

No value is returned.

spl_autoload_extensions

spl_autoload_extensions -- Register and return default file extensions for spl_autoload

Description

string **spl_autoload_extensions** ([string `$file_extensions`])

This function can modify and check the file extensions that the built in **__autoload()** fallback function spl_autoload() will be using.

Parameters

`file_extensions`
: When calling without an argument, it simply returns the current list of extensions each separated by comma. To modify the list of file extensions, simply invoke the functions with the new list of file extensions to use in a single string with each extensions separated by comma.

Return Values

A comma delimitated list of default file extensions for spl_autoload().

spl_autoload_functions

spl_autoload_functions -- Return all registered __autoload() functions

Description

array **spl_autoload_functions** (void)

Get all registered __autoload() functions.

Parameters

This function has no parameters.

Return Values

An array of all registered __autoload functions. If the autoload stack is not activated then the return value is **FALSE**. If no function is registered the return value will be an empty array.

spl_autoload_register

spl_autoload_register -- Register given function as __autoload() implementation

Description

bool **spl_autoload_register** ([callback `$autoload_function`])

Register a function with the spl provided __autoload stack. If the stack is not yet activated it will be activated.

If your code has an existing __autoload function then this function must be explicitly registered on the __autoload stack. This is because spl_autoload_register() will effectively replace the engine cache for the __autoload function by either spl_autoload() or spl_autoload_call().

Parameters

`autoload_function`
The autoload function being registered. If no parameter is provided, then the default implementation of spl_autoload() will be registered.

Return Values

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

Examples

Example #3 - spl_autoload_register() example

```
<?php

namespace Foobar;

class Foo {
   static public function test($name) {
       print '[['. $name .']]';
   }
}

spl_autoload_register(__NAMESPACE__ .'::Foo::test'); // As of PHP 5.3.0

new InexistentClass;

?>
```

The above example will output something similar to:

```
[[Foobar::InexistentClass]]
Fatal error: Class 'Foobar::InexistentClass' not found in ...
```

spl_autoload_unregister

spl_autoload_unregister -- Unregister given function as __autoload() implementation

Description

bool **spl_autoload_unregister** (mixed `$autoload_function`)

Unregister a function from the spl provided __autoload stack. If the stack is activated and empty after unregistering the given function then it will be deactivated.

When this function results in the autoload stack being activated an existing __autoload function will not be reactivated.

Parameters

`autoload_function`
The autoload function being unregistered.

Return Values

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

spl_autoload

spl_autoload -- Default implementation for __autoload()

Description

void **spl_autoload** (string `$class_name` [, string `$file_extensions`])

This function is intended to be used as a default implementation for __autoload(). If nothing else is specified and autoload_register() is called without any parameters then this functions will be used for any later call to __autoload().

Parameters

`class_name`

`file_extensions`
By default it checks all include paths to contain filenames built up by the lowercase class name appended by the filename extensions .inc and .php.

Return Values

No value is returned.

spl_classes

spl_classes -- Return available SPL classes

Description

array **spl_classes** (void)

This function returns an array with the current available SPL classes.

Examples

Example #4 - spl_classes() example

```
<?php

print_r(spl_classes());

?>
```

The above example will output something similar to:

```
Array
(
   [ArrayObject] => ArrayObject
   [ArrayIterator] => ArrayIterator
   [CachingIterator] => CachingIterator
   [RecursiveCachingIterator] => RecursiveCachingIterator
   [DirectoryIterator] => DirectoryIterator
   [FilterIterator] => FilterIterator
   [LimitIterator] => LimitIterator
   [ParentIterator] => ParentIterator
   [RecursiveDirectoryIterator] => RecursiveDirectoryIterator
   [RecursiveIterator] => RecursiveIterator
   [RecursiveIteratorIterator] => RecursiveIteratorIterator
   [SeekableIterator] => SeekableIterator
   [SimpleXMLIterator] => SimpleXMLIterator
)
```

spl_object_hash

spl_object_hash -- Return hash id for given object

Description

string **spl_object_hash** (object `$obj`)

This function returns an unique identifier for the object. This id can be used as a hash key for storing objects or for identifying an object.

Parameters

`object`
Any object.

Return Values

A string that is unique for each object and is always the same for the same object.

Examples

Example #5 - A spl_object_hash() example

```
<?php
$id = spl_object_hash($object);
$storage[$id] = $object;
?>
```

The ArrayIterator class

Introduction

This iterator allows to unset and modify values and keys while iterating over Arrays and Objects.

When you want to iterate over the same array multiple times you need to instanciate ArrayObject and let it create ArrayIterator instances that refer to it either by using foreach or by calling its getIterator() method manually.

Class synopsis

ArrayIterator

ArrayIterator implements Iterator, Traversable, ArrayAccess, SeekableIterator, Countable {

/* Methods */

mixed **ArrayIterator::current** (void)

mixed **ArrayIterator::key** (void)

void **ArrayIterator::next** (void)

void **ArrayIterator::rewind** (void)

void **ArrayIterator::seek** (int `$position`)

bool **ArrayIterator::valid** (void)

}

ArrayIterator::current

ArrayIterator::current -- Return current array entry

Description

mixed **ArrayIterator::current** (void)

Get the current array entry.

Parameters

This function has no parameters.

Return Values

The current array entry.

Examples

Example #6 - ArrayIterator::current() example

```
<?php
$array = array('1' => 'one',
               '2' => 'two',
               '3' => 'three');

$arrayobject = new ArrayObject($array);

for($iterator = $arrayobject->getIterator();
   $iterator->valid();
   $iterator->next()) {

   echo $iterator->key() . ' => ' . $iterator->current() . "\n";
}
?>
```

The above example will output:

```
1 => one
2 => two
3 => three
```

ArrayIterator::key

ArrayIterator::key -- Return current array key

Description

mixed **ArrayIterator::key** (void)

This function returns the current array key

Parameters

This function has no parameters.

Return Values

The current array key.

Examples

Example #7 - ArrayIterator::key() example

```
<?php
$array = array('key' => 'value');

$arrayobject = new ArrayObject($array);
$iterator = $arrayobject->getIterator();

echo $iterator->key(); //key
?>
```

ArrayIterator::next

ArrayIterator::next -- Move to next entry

Description

void **ArrayIterator::next** (void)

The iterator to the next entry.

Parameters

This function has no parameters.

Return Values

No value is returned.

Examples

Example #8 - ArrayIterator::next() example

```
<?php
$arrayobject = new ArrayObject();

$arrayobject[] = 'zero';
$arrayobject[] = 'one';

$iterator = $arrayobject->getIterator();

while($iterator->valid()) {
   echo $iterator->key() . ' => ' . $iterator->current() . "\n";

   $iterator->next();
}
?>
```

The above example will output:

```
0 => zero
1 => one
```

ArrayIterator::rewind

ArrayIterator::rewind -- Rewind array back to the start

Description

void **ArrayIterator::rewind** (void)

This rewinds the iterator to the beginning.

Parameters

This function has no parameters.

Return Values

No value is returned.

Examples

Example #9 - ArrayIterator::rewind() example

```
<?php
$arrayobject = new ArrayObject();

$arrayobject[] = 'zero';
$arrayobject[] = 'one';
$arrayobject[] = 'two';

$iterator = $arrayobject->getIterator();

$iterator->next();
echo $iterator->key(); //1

$iterator->rewind(); //rewinding to the begining
echo $iterator->key(); //0
?>
```

ArrayIterator::seek

ArrayIterator::seek -- Seek to position

Description

void **ArrayIterator::seek** (int `$position`)

Warning
This function is currently not documented; only its argument list is available.

Parameters

`position`
The position to seek to.

Return Values

No value is returned.

ArrayIterator::valid

ArrayIterator::valid -- Check whether array contains more entries

Description

bool **ArrayIterator::valid** (void)

Checks if the array contains any more entries.

Parameters

This function has no parameters.

Return Values

No value is returned.

Examples

Example #10 - ArrayIterator::valid() example

```
<?php
$array = array('1' => 'one');

$arrayobject = new ArrayObject($array);
$iterator = $arrayobject->getIterator();

var_dump($iterator->valid()); //bool(true)

$iterator->next(); // advance to the next item

//bool(false) because there is only one array element
var_dump($iterator->valid());
?>
```

The ArrayObject class

Introduction

...

Class synopsis

ArrayObject

ArrayObject implements IteratorAggregate, Traversable, ArrayAccess, Countable {

/* Methods */

ArrayObject::__construct (mixed `$input`)

void **ArrayObject::append** (mixed `$newval`)

int **ArrayObject::count** (void)

ArrayIterator **ArrayObject::getIterator** (void)

bool **ArrayObject::offsetExists** (mixed `$index`)

mixed **ArrayObject::offsetGet** (mixed `$index`)

void **ArrayObject::offsetSet** (mixed `$index`, mixed `$newval`)

void **ArrayObject::offsetUnset** (mixed `$index`)

}

ArrayObject::append

ArrayObject::append -- Appends the value

Description

void **ArrayObject::append** (mixed `$newval`)

Warning
This function is currently not documented; only its argument list is available.

Parameters

This function has no parameters.

Return Values

No value is returned.

ArrayObject::__construct

ArrayObject::__construct -- Construct a new array object

Description

ArrayObject::__construct (mixed `$input`)

This constructs a new array object.

Parameters

`input`
The *`input`* parameter accepts an array or another ArrayObject.

Return Values

No value is returned.

Examples

Example #11 - ArrayObject::__construct() example

```
<?php
$array = array('1' => 'one',
               '2' => 'two',
               '3' => 'three');

$arrayobject = new ArrayObject($array);

var_dump($arrayobject);
?>
```

The above example will output:

```
object(ArrayObject)#1 (3) {
 [1]=>
 string(3) "one"
 [2]=>
 string(3) "two"
 [3]=>
 string(5) "three"
}
```

ArrayObject::count

ArrayObject::count -- Get the number of elements in the Iterator

Description

int **ArrayObject::count** (void)

Warning
This function is currently not documented; only its argument list is available.

Parameters

This function has no parameters.

Return Values

The number of elements in the Iterator.

ArrayObject::getIterator

ArrayObject::getIterator -- Create a new iterator from an ArrayObject instance

Description

ArrayIterator **ArrayObject::getIterator** (void)

Create a new iterator from an ArrayObject instance.

Parameters

This function has no parameters.

Return Values

An iterator from an ArrayObject.

Examples

Example #12 - ArrayObject::getIterator() example

```
<?php
$array = array('1' => 'one',
               '2' => 'two',
               '3' => 'three');

$arrayobject = new ArrayObject($array);

$iterator = $arrayobject->getIterator();

while($iterator->valid()) {
   echo $iterator->key() . ' => ' . $iterator->current() . "\n";

   $iterator->next();
}
?>
```

The above example will output:

```
1 => one
2 => two
3 => three
```

ArrayObject::offsetExists

ArrayObject::offsetExists -- Returns whether the requested $index exists

Description

bool **ArrayObject::offsetExists** (mixed `$index`)

Warning
This function is currently not documented; only its argument list is available.

Parameters

`index`
The index being checked.

Return Values

TRUE if the requested $index exists, otherwise **FALSE**

ArrayObject::offsetGet

ArrayObject::offsetGet -- Returns the value at the specified $index

Description

mixed **ArrayObject::offsetGet** (mixed `$index`)

Warning
This function is currently not documented; only its argument list is available.

Parameters

`index`
The index with the value.

Return Values

The value at the specified $index.

ArrayObject::offsetSet

ArrayObject::offsetSet -- Sets the value at the specified $index to $newval

Description

void **ArrayObject::offsetSet** (mixed `$index`, mixed `$newval`)

Warning
This function is currently not documented; only its argument list is available.

Sets the value at the specified index to newval.

Parameters

`index`
: The index being set.

`newval`
: The new value for the *`index`*.

Return Values

No value is returned.

ArrayObject::offsetUnset

ArrayObject::offsetUnset -- Unsets the value at the specified $index

Description

void **ArrayObject::offsetUnset** (mixed `$index`)

Warning
This function is currently not documented; only its argument list is available.

Unsets the value at the specified index.

Parameters

`index`
The index being unset.

Return Values

No value is returned.

The CachingIterator class

Introduction

...

Class synopsis

CachingIterator

CachingIterator extends IteratorIterator implements OuterIterator, Traversable, Iterator, ArrayAccess, Countable {

/* Methods */

bool **CachingIterator::hasNext** (void)

void **CachingIterator::next** (void)

void **CachingIterator::rewind** (void)

string **CachingIterator::__toString** (void)

bool **CachingIterator::valid** (void)

}

CachingIterator::hasNext

CachingIterator::hasNext -- Check whether the inner iterator has a valid next element

Description

bool **CachingIterator::hasNext** (void)

Warning
This function is currently not documented; only its argument list is available.

Parameters

This function has no parameters.

Return Values

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

CachingIterator::next

CachingIterator::next -- Move the iterator forward

Description

void **CachingIterator::next** (void)

Warning
This function is currently not documented; only its argument list is available.

Move the iterator forward.

Parameters

This function has no parameters.

Return Values

No value is returned.

CachingIterator::rewind

CachingIterator::rewind -- Rewind the iterator

Description

void **CachingIterator::rewind** (void)

Warning
This function is currently not documented; only its argument list is available.

Rewind the iterator.

Parameters

This function has no parameters.

Return Values

No value is returned.

CachingIterator::__toString

CachingIterator::__toString -- Return the string representation of the current element

Description

string **CachingIterator::__toString** (void)

Warning
This function is currently not documented; only its argument list is available.

Get the string representation of the current element.

Parameters

This function has no parameters.

Return Values

The string representation of the current element.

CachingIterator::valid

CachingIterator::valid -- Check whether the current element is valid

Description

bool **CachingIterator::valid** (void)

Warning
This function is currently not documented; only its argument list is available.

Check whether the current element is valid.

Parameters

This function has no parameters.

Return Values

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

The RecursiveCachingIterator class

Introduction

...

Class synopsis

RecursiveCachingIterator

RecursiveCachingIterator extends CachingIterator implements Countable, ArrayAccess, Iterator, Traversable, OuterIterator, RecursiveIterator {

/* Methods */

RecursiveCachingIterator **RecursiveCachingIterator::getChildren** (void)

boolean **RecursiveCachingIterator::hasChildren** (void)

/* Inherits */

bool **CachingIterator::hasNext** (void)

void **CachingIterator::next** (void)

void **CachingIterator::rewind** (void)

string **CachingIterator::__toString** (void)

bool **CachingIterator::valid** (void)

}

RecursiveCachingIterator::getChildren

RecursiveCachingIterator::getChildren -- Return the inner iterator's children as a RecursiveCachingIterator

Description

RecursiveCachingIterator **RecursiveCachingIterator::getChildren** (void)

Warning
This function is currently not documented; only its argument list is available.

Parameters

This function has no parameters.

Return Values

The inner iterator's children, as a RecursiveCachingIterator.

RecursiveCachingIterator::hasChildren

RecursiveCachingIterator::hasChildren -- Check whether the current element of the inner iterator has children

Description

boolean **RecursiveCachingIterator::hasChildren** (void)

Warning
This function is currently not documented; only its argument list is available.

Parameters

This function has no parameters.

Return Values

TRUE if the inner iterator has children, otherwise **FALSE**

The DirectoryIterator class

Introduction

...

Class synopsis

DirectoryIterator

DirectoryIterator extends SplFileInfo implements Iterator, Traversable {

/* Methods */

DirectoryIterator::__construct (string `$path`)

DirectoryIterator **DirectoryIterator::current** (void)

int **DirectoryIterator::getATime** (void)

int **DirectoryIterator::getCTime** (void)

string **DirectoryIterator::getFilename** (void)

int **DirectoryIterator::getGroup** (void)

int **DirectoryIterator::getInode** (void)

int **DirectoryIterator::getMTime** (void)

int **DirectoryIterator::getOwner** (void)

string **DirectoryIterator::getPath** (void)

string **DirectoryIterator::getPathname** (void)

int **DirectoryIterator::getPerms** (void)

int **DirectoryIterator::getSize** (void)

string **DirectoryIterator::getType** (void)

bool **DirectoryIterator::isDir** (void)

bool **DirectoryIterator::isDot** (void)

bool **DirectoryIterator::isExecutable** (void)

bool **DirectoryIterator::isFile** (void)

bool **DirectoryIterator::isLink** (void)

bool **DirectoryIterator::isReadable** (void)

bool **DirectoryIterator::isWritable** (void)

string **DirectoryIterator::key** (void)

void **DirectoryIterator::next** (void)

void **DirectoryIterator::rewind** (void)

string **DirectoryIterator::valid** (void)
}

DirectoryIterator::__construct

DirectoryIterator::__construct -- Constructs a new dir iterator from a path

Description

DirectoryIterator::__construct (string `$path`)

Warning
This function is currently not documented; only its argument list is available.

Constructs a new dir iterator from a path.

Parameters

`path`
The path.

Return Values

No value is returned.

DirectoryIterator::current

DirectoryIterator::current -- Return this (needed for Iterator interface)

Description

DirectoryIterator **DirectoryIterator::current** (void)

Warning
This function is currently not documented; only its argument list is available.

Parameters

This function has no parameters.

Return Values

Return this (needed for Iterator interface).

DirectoryIterator::getATime

DirectoryIterator::getATime -- Get last access time of file

Description

int **DirectoryIterator::getATime** (void)

Warning
This function is currently not documented; only its argument list is available.

Get the last access time of file.

Parameters

This function has no parameters.

Return Values

The last access time of the file.

DirectoryIterator::getCTime

DirectoryIterator::getCTime -- Get inode modification time of file

Description

int **DirectoryIterator::getCTime** (void)

Warning
This function is currently not documented; only its argument list is available.

Get the inode modification time of file.

Parameters

This function has no parameters.

Return Values

The last modification time of the file.

DirectoryIterator::getFilename

DirectoryIterator::getFilename -- Return filename of current dir entry

Description

string **DirectoryIterator::getFilename** (void)

Warning
This function is currently not documented; only its argument list is available.

Get the filename of the current dir entry.

Parameters

This function has no parameters.

Return Values

The filename of the current dir entry.

DirectoryIterator::getGroup

DirectoryIterator::getGroup -- Get file group

Description

int **DirectoryIterator::getGroup** (void)

Warning
This function is currently not documented; only its argument list is available.

Get the files group.

Parameters

This function has no parameters.

Return Values

The file group of the file.

DirectoryIterator::getInode

DirectoryIterator::getInode -- Get file inode

Description

int **DirectoryIterator::getInode** (void)

Warning
This function is currently not documented; only its argument list is available.

Get the inode of the file.

Parameters

This function has no parameters.

Return Values

The inode of the file.

DirectoryIterator::getMTime

DirectoryIterator::getMTime -- Get last modification time of file

Description

int **DirectoryIterator::getMTime** (void)

Warning
This function is currently not documented; only its argument list is available.

Get the last modification time of the file.

Parameters

This function has no parameters.

Return Values

The last modification time of the file.

DirectoryIterator::getOwner

DirectoryIterator::getOwner -- Get file owner

Description

int **DirectoryIterator::getOwner** (void)

Warning
This function is currently not documented; only its argument list is available.

Get the owner of the file.

Parameters

This function has no parameters.

Return Values

The file owner of the file.

DirectoryIterator::getPath

DirectoryIterator::getPath -- Return directory path

Description

string **DirectoryIterator::getPath** (void)

Warning
This function is currently not documented; only its argument list is available.

Get the directory path.

Parameters

This function has no parameters.

Return Values

The directory path.

DirectoryIterator::getPathname

DirectoryIterator::getPathname -- Return path and filename of current dir entry

Description

string **DirectoryIterator::getPathname** (void)

Warning
This function is currently not documented; only its argument list is available.

Get the path and filename of the current dir entry.

Parameters

This function has no parameters.

Return Values

The path and filename of current dir entry.

DirectoryIterator::getPerms

DirectoryIterator::getPerms -- Get file permissions

Description

int **DirectoryIterator::getPerms** (void)

Warning
This function is currently not documented; only its argument list is available.

Get the permissions of the file.

Parameters

This function has no parameters.

Return Values

The file permissions of the file.

DirectoryIterator::getSize

DirectoryIterator::getSize -- Get file size

Description

int **DirectoryIterator::getSize** (void)

Warning
This function is currently not documented; only its argument list is available.

Get the file size.

Parameters

This function has no parameters.

Return Values

The size of the file.

DirectoryIterator::getType

DirectoryIterator::getType -- Get file type

Description

string **DirectoryIterator::getType** (void)

Warning
This function is currently not documented; only its argument list is available.

Get the file type.

Parameters

This function has no parameters.

Return Values

The type of the file.

DirectoryIterator::isDir

DirectoryIterator::isDir -- Returns true if file is directory

Description

bool **DirectoryIterator::isDir** (void)

Warning
This function is currently not documented; only its argument list is available.

Check if the file is a directory.

Parameters

This function has no parameters.

Return Values

TRUE if it is a directory, otherwise **FALSE**

DirectoryIterator::isDot

DirectoryIterator::isDot -- Returns true if current entry is '.' or '..'

Description

bool **DirectoryIterator::isDot** (void)

Warning
This function is currently not documented; only its argument list is available.

Check whether it's a directory and either. or...

Parameters

This function has no parameters.

Return Values

TRUE if the entry is. or.., otherwise **FALSE**

DirectoryIterator::isExecutable

DirectoryIterator::isExecutable -- Returns true if file is executable

Description

bool **DirectoryIterator::isExecutable** (void)

Warning
This function is currently not documented; only its argument list is available.

Checks if the file is executable.

Parameters

This function has no parameters.

Return Values

TRUE if the entry is executable, otherwise **FALSE**

DirectoryIterator::isFile

DirectoryIterator::isFile -- Returns true if file is a regular file

Description

bool **DirectoryIterator::isFile** (void)

Warning
This function is currently not documented; only its argument list is available.

Checks if it's a regular file.

Parameters

This function has no parameters.

Return Values

TRUE if the entry is a regular file, otherwise **FALSE**

DirectoryIterator::isLink

DirectoryIterator::isLink -- Returns true if file is symbolic link

Description

bool **DirectoryIterator::isLink** (void)

Warning
This function is currently not documented; only its argument list is available.

Checks if the entry is a symbolic link.

Parameters

This function has no parameters.

Return Values

TRUE if the entry is a symbolic link, otherwise **FALSE**

DirectoryIterator::isReadable

DirectoryIterator::isReadable -- Returns true if file can be read

Description

bool **DirectoryIterator::isReadable** (void)

Warning
This function is currently not documented; only its argument list is available.

Checks if the entry is readable.

Parameters

This function has no parameters.

Return Values

TRUE if the file is readable, otherwise **FALSE**

DirectoryIterator::isWritable

DirectoryIterator::isWritable -- Returns true if file can be written

Description

bool **DirectoryIterator::isWritable** (void)

Warning
This function is currently not documented; only its argument list is available.

Checks if the entry is writable.

Parameters

This function has no parameters.

Return Values

TRUE if the file is writable, otherwise **FALSE**

DirectoryIterator::key

DirectoryIterator::key -- Return current dir entry

Description

string **DirectoryIterator::key** (void)

Warning
This function is currently not documented; only its argument list is available.

Get the current dir entry.

Parameters

This function has no parameters.

Return Values

The current dir entry.

DirectoryIterator::next

DirectoryIterator::next -- Move to next entry

Description

void **DirectoryIterator::next** (void)

Warning
This function is currently not documented; only its argument list is available.

Move to the next entry.

Parameters

This function has no parameters.

Return Values

No value is returned.

DirectoryIterator::rewind

DirectoryIterator::rewind -- Rewind dir back to the start

Description

void **DirectoryIterator::rewind** (void)

Warning
This function is currently not documented; only its argument list is available.

Rewind dir back to the start.

Parameters

This function has no parameters.

Return Values

No value is returned.

DirectoryIterator::valid

DirectoryIterator::valid -- Check whether dir contains more entries

Description

string **DirectoryIterator::valid** (void)

Warning
This function is currently not documented; only its argument list is available.

Check whether dir contains more entries.

Parameters

This function has no parameters.

Return Values

No value is returned.

The FilterIterator class

Introduction

...

Class synopsis

FilterIterator

abstract FilterIterator extends IteratorIterator implements OuterIterator, Traversable, Iterator {

    /* Methods */

    mixed **FilterIterator::current** (void)

    Iterator **FilterIterator::getInnerIterator** (void)

    mixed **FilterIterator::key** (void)

    void **FilterIterator::next** (void)

    void **FilterIterator::rewind** (void)

    bool **FilterIterator::valid** (void)
}

FilterIterator::current

FilterIterator::current -- Get the current element value

Description

mixed **FilterIterator::current** (void)

Warning
This function is currently not documented; only its argument list is available.

Get the current element value.

Parameters

This function has no parameters.

Return Values

The current element value.

See Also

- FilterIterator::key()
- FilterIterator::next()

FilterIterator::getInnerIterator

FilterIterator::getInnerIterator -- Get the inner iterator

Description

Iterator **FilterIterator::getInnerIterator** (void)

Warning
This function is currently not documented; only its argument list is available.

Get the inner iterator.

Parameters

This function has no parameters.

Return Values

The inner iterator.

FilterIterator::key

FilterIterator::key -- Get the current key

Description

mixed **FilterIterator::key** (void)

Warning
This function is currently not documented; only its argument list is available.

Get the current key.

Parameters

This function has no parameters.

Return Values

The current key.

See Also

- FilterIterator::next()
- FilterIterator::current()

FilterIterator::next

FilterIterator::next -- Move the iterator forward

Description

void **FilterIterator::next** (void)

Warning
This function is currently not documented; only its argument list is available.

Move the iterator forward.

Parameters

This function has no parameters.

Return Values

No value is returned.

See Also

- FilterIterator::current()
- FilterIterator::next()

FilterIterator::rewind

FilterIterator::rewind -- Rewind the iterator

Description

void **FilterIterator::rewind** (void)

Warning
This function is currently not documented; only its argument list is available.

Rewind the iterator.

Parameters

This function has no parameters.

Return Values

No value is returned.

See Also

- FilterIterator::current()
- FilterIterator::key()
- FilterIterator::next()

FilterIterator::valid

FilterIterator::valid -- Check whether the current element is valid

Description

bool **FilterIterator::valid** (void)

Warning
This function is currently not documented; only its argument list is available.

Checks whether the current element is valid.

Parameters

This function has no parameters.

Return Values

TRUE if the current element is valid, otherwise **FALSE**

The LimitIterator class

Introduction

...

Class synopsis

LimitIterator

LimitIterator extends IteratorIterator implements OuterIterator, Traversable, Iterator {

    /* Methods */

    int **LimitIterator::getPosition** (void)

    void **LimitIterator::next** (void)

    void **LimitIterator::rewind** (void)

    void **LimitIterator::seek** (int `$position`)

    bool **LimitIterator::valid** (void)
}

LimitIterator::getPosition

LimitIterator::getPosition -- Return the current position

Description

int **LimitIterator::getPosition** (void)

Warning
This function is currently not documented; only its argument list is available.

Parameters

This function has no parameters.

Return Values

The current position.

LimitIterator::next

LimitIterator::next -- Move the iterator forward

Description

void **LimitIterator::next** (void)

Moves the iterator forward.

Warning
This function is currently not documented; only its argument list is available.

Parameters

This function has no parameters.

Return Values

No value is returned.

LimitIterator::rewind

LimitIterator::rewind -- Rewind the iterator to the specified starting offset

Description

void **LimitIterator::rewind** (void)

Rewinds the iterator to the specified starting offset.

Warning
This function is currently not documented; only its argument list is available.

Parameters

This function has no parameters.

Return Values

No value is returned.

LimitIterator::seek

LimitIterator::seek -- Seek to the given position

Description

void **LimitIterator::seek** (int `$position`)

Warning
This function is currently not documented; only its argument list is available.

Parameters

`position`
: The position being seeked to.

Return Values

No value is returned.

LimitIterator::valid

LimitIterator::valid -- Check whether the current element is valid

Description

bool **LimitIterator::valid** (void)

Checks whether the current element is valid.

Warning
This function is currently not documented; only its argument list is available.

Parameters

This function has no parameters.

Return Values

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

The ParentIterator class

Introduction

...

Class synopsis

ParentIterator

ParentIterator extends RecursiveFilterIterator implements RecursiveIterator, OuterIterator, Traversable, Iterator {

    /* Methods */

    ParentIterator **ParentIterator::getChildren** (void)

    bool **ParentIterator::hasChildren** (void)

    void **ParentIterator::next** (void)

    void **ParentIterator::rewind** (void)
}

ParentIterator::getChildren

ParentIterator::getChildren -- Return the inner iterator's children contained in a ParentIterator

Description

ParentIterator **ParentIterator::getChildren** (void)

Get the the inner iterator's children contained in a ParentIterator.

Warning
This function is currently not documented; only its argument list is available.

Parameters

This function has no parameters.

Return Values

An object.

ParentIterator::hasChildren

ParentIterator::hasChildren -- Check whether the inner iterator's current element has children

Description

bool **ParentIterator::hasChildren** (void)

Check whether the inner iterator's current element has children.

Warning
This function is currently not documented; only its argument list is available.

Parameters

This function has no parameters.

Return Values

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

ParentIterator::next

ParentIterator::next -- Move the iterator forward

Description

void **ParentIterator::next** (void)

Moves the iterator forward.

Warning
This function is currently not documented; only its argument list is available.

Parameters

This function has no parameters.

Return Values

No value is returned.

ParentIterator::rewind

ParentIterator::rewind -- Rewind the iterator

Description

void **ParentIterator::rewind** (void)

Rewinds the iterator.

Warning
This function is currently not documented; only its argument list is available.

Parameters

This function has no parameters.

Return Values

No value is returned.

The RecursiveDirectoryIterator class

Introduction

...

Class synopsis

RecursiveDirectoryIterator

RecursiveDirectoryIterator extends DirectoryIterator implements Traversable, Iterator, RecursiveIterator {

/* Methods */

object **RecursiveDirectoryIterator::getChildren** (void)

bool **RecursiveDirectoryIterator::hasChildren** ([bool `$allow_links`])

string **RecursiveDirectoryIterator::key** (void)

void **RecursiveDirectoryIterator::next** (void)

void **RecursiveDirectoryIterator::rewind** (void)

/* Inherits */

DirectoryIterator **DirectoryIterator::current** (void)

int **DirectoryIterator::getATime** (void)

int **DirectoryIterator::getCTime** (void)

string **DirectoryIterator::getFilename** (void)

int **DirectoryIterator::getGroup** (void)

int **DirectoryIterator::getInode** (void)

int **DirectoryIterator::getMTime** (void)

int **DirectoryIterator::getOwner** (void)

```
    string DirectoryIterator::getPath ( void )

    string DirectoryIterator::getPathname ( void )

    int DirectoryIterator::getPerms ( void )

    int DirectoryIterator::getSize ( void )

    string DirectoryIterator::getType ( void )

    bool DirectoryIterator::isDir ( void )

    bool DirectoryIterator::isDot ( void )

    bool DirectoryIterator::isExecutable ( void )

    bool DirectoryIterator::isFile ( void )

    bool DirectoryIterator::isLink ( void )

    bool DirectoryIterator::isReadable ( void )

    bool DirectoryIterator::isWritable ( void )

    string DirectoryIterator::key ( void )

    void DirectoryIterator::next ( void )

    void DirectoryIterator::rewind ( void )

    string DirectoryIterator::valid ( void )
}
```

RecursiveDirectoryIterator::getChildren

RecursiveDirectoryIterator::getChildren -- Returns an iterator for the current entry if it is a directory

Description

object **RecursiveDirectoryIterator::getChildren** (void)

Warning
This function is currently not documented; only its argument list is available.

Parameters

This function has no parameters.

Return Values

An iterator for the current entry, if it is a directory.

RecursiveDirectoryIterator::hasChildren

RecursiveDirectoryIterator::hasChildren -- Returns whether current entry is a directory and not '.' or '..'

Description

bool **RecursiveDirectoryIterator::hasChildren** ([bool `$allow_links`])

Warning
This function is currently not documented; only its argument list is available.

Parameters

`allow_links`

Return Values

Returns whether the current entry is a directory, but not '.' or '..'

RecursiveDirectoryIterator::key

RecursiveDirectoryIterator::key -- Return path and filename of current dir entry

Description

string **RecursiveDirectoryIterator::key** (void)

Warning
This function is currently not documented; only its argument list is available.

Parameters

This function has no parameters.

Return Values

The path and filename of the current dir entry.

RecursiveDirectoryIterator::next

RecursiveDirectoryIterator::next -- Move to next entry

Description

void **RecursiveDirectoryIterator::next** (void)

Warning
This function is currently not documented; only its argument list is available.

Parameters

This function has no parameters.

Return Values

No value is returned.

RecursiveDirectoryIterator::rewind

RecursiveDirectoryIterator::rewind -- Rewind dir back to the start

Description

void **RecursiveDirectoryIterator::rewind** (void)

Warning
This function is currently not documented; only its argument list is available.

Parameters

This function has no parameters.

Return Values

No value is returned.

The RecursiveIteratorIterator class

Introduction

...

Class synopsis

RecursiveIteratorIterator

```
RecursiveIteratorIterator implements OuterIterator, Traversable, Iterator {
    /*  Methods */
    mixed RecursiveIteratorIterator::current ( void )
    int RecursiveIteratorIterator::getDepth ( void )
    RecursiveIterator RecursiveIteratorIterator::getSubIterator ( void )
    mixed RecursiveIteratorIterator::key ( void )
    void RecursiveIteratorIterator::next ( void )
    void RecursiveIteratorIterator::rewind ( void )
    boolean RecursiveIteratorIterator::valid ( void )
}
```

RecursiveIteratorIterator::current

RecursiveIteratorIterator::current -- Access the current element value

Description

mixed **RecursiveIteratorIterator::current** (void)

Warning
This function is currently not documented; only its argument list is available.

Parameters

This function has no parameters.

Return Values

The current elements value.

RecursiveIteratorIterator::getDepth

RecursiveIteratorIterator::getDepth -- Get the current depth of the recursive iteration

Description

int **RecursiveIteratorIterator::getDepth** (void)

Warning
This function is currently not documented; only its argument list is available.

Parameters

This function has no parameters.

Return Values

The current depth of the recursive iteration.

RecursiveIteratorIterator::getSubIterator

RecursiveIteratorIterator::getSubIterator -- The current active sub iterator

Description

RecursiveIterator **RecursiveIteratorIterator::getSubIterator** (void)

Warning
This function is currently not documented; only its argument list is available.

Parameters

This function has no parameters.

Return Values

The current active sub iterator.

RecursiveIteratorIterator::key

RecursiveIteratorIterator::key -- Access the current key

Description

mixed **RecursiveIteratorIterator::key** (void)

Warning
This function is currently not documented; only its argument list is available.

Parameters

This function has no parameters.

Return Values

The current key.

RecursiveIteratorIterator::next

RecursiveIteratorIterator::next -- Move forward to the next element

Description

void **RecursiveIteratorIterator::next** (void)

Warning
This function is currently not documented; only its argument list is available.

Parameters

This function has no parameters.

Return Values

No value is returned.

RecursiveIteratorIterator::rewind

RecursiveIteratorIterator::rewind -- Rewind the iterator to the first element of the top level inner iterator

Description

void **RecursiveIteratorIterator::rewind** (void)

Warning
This function is currently not documented; only its argument list is available.

Parameters

This function has no parameters.

Return Values

No value is returned.

RecursiveIteratorIterator::valid

RecursiveIteratorIterator::valid -- Check whether the current position is valid

Description

boolean **RecursiveIteratorIterator::valid** (void)

Warning
This function is currently not documented; only its argument list is available.

Parameters

This function has no parameters.

Return Values

TRUE if the current position is valid, otherwise **FALSE**

The SimpleXMLIterator class

Introduction

...

Class synopsis

SimpleXMLIterator

SimpleXMLIterator extends SimpleXMLElement implements RecursiveIterator, Traversable, Iterator, Countable {

/* Methods */

mixed **SimpleXMLIterator::current** (void)

object **SimpleXMLIterator::getChildren** (void)

bool **SimpleXMLIterator::hasChildren** (void)

mixed **SimpleXMLIterator::key** (void)

void **SimpleXMLIterator::next** (void)

void **SimpleXMLIterator::rewind** (void)

bool **SimpleXMLIterator::valid** (void)
}

SimpleXMLIterator::current

SimpleXMLIterator::current -- Return current SimpleXML entry

Description

mixed **SimpleXMLIterator::current** (void)

Warning
This function is currently not documented; only its argument list is available.

Parameters

This function has no parameters.

Return Values

The current SimpleXML entry.

SimpleXMLIterator::getChildren

SimpleXMLIterator::getChildren -- Returns an iterator for the current entry if it is a SimpleXML object

Description

object **SimpleXMLIterator::getChildren** (void)

Warning
This function is currently not documented; only its argument list is available.

Parameters

This function has no parameters.

Return Values

An iterator for the current entry, if it is a SimpleXML object.

SimpleXMLIterator::hasChildren

SimpleXMLIterator::hasChildren -- Returns whether current entry is a SimpleXML object

Description

bool **SimpleXMLIterator::hasChildren** (void)

Warning
This function is currently not documented; only its argument list is available.

Parameters

This function has no parameters.

Return Values

TRUE if the current entry is a SimpleXML object, otherwise **FALSE**

SimpleXMLIterator::key

SimpleXMLIterator::key -- Return current SimpleXML key

Description

mixed **SimpleXMLIterator::key** (void)

Warning
This function is currently not documented; only its argument list is available.

Parameters

This function has no parameters.

Return Values

The current SimpleXML key.

SimpleXMLIterator::next

SimpleXMLIterator::next -- Move to next entry

Description

void **SimpleXMLIterator::next** (void)

Warning
This function is currently not documented; only its argument list is available.

Parameters

This function has no parameters.

Return Values

No value is returned.

SimpleXMLIterator::rewind

SimpleXMLIterator::rewind -- Rewind SimpleXML back to the start

Description

void **SimpleXMLIterator::rewind** (void)

Warning
This function is currently not documented; only its argument list is available.

Parameters

This function has no parameters.

Return Values

No value is returned.

SimpleXMLIterator::valid

SimpleXMLIterator::valid -- Check whether SimpleXML contains more entries

Description

bool **SimpleXMLIterator::valid** (void)

Warning
This function is currently not documented; only its argument list is available.

Parameters

This function has no parameters.

Return Values

TRUE if contains more SimpleXML entries, otherwise **FALSE**

The SplDoublyLinkedList class

Introduction

The SplDoublyLinkedList class provides the main functionalities of a doubly linked list.

Class synopsis

SplDoublyLinkedList

SplDoublyLinkedList implements Iterator, ArrayAccess, Countable {

/* Methods */

SplDoublyLinkedList::__construct (void)

mixed **SplDoublyLinkedList::bottom** (void)

int **SplDoublyLinkedList::count** (void)

mixed **SplDoublyLinkedList::current** (void)

int **SplDoublyLinkedList::getIteratorMode** (void)

bool **SplDoublyLinkedList::isEmpty** (void)

mixed **SplDoublyLinkedList::key** (void)

void **SplDoublyLinkedList::next** (void)

bool **SplDoublyLinkedList::offsetExists** (mixed `$index`)

mixed **SplDoublyLinkedList::offsetGet** (mixed `$index`)

void **SplDoublyLinkedList::offsetSet** (mixed `$index`, mixed `$newval`)

void **SplDoublyLinkedList::offsetUnset** (mixed `$index`)

mixed **SplDoublyLinkedList::pop** (void)

void **SplDoublyLinkedList::push** (mixed `$value`)

void **SplDoublyLinkedList::rewind** (void)

```
    void SplDoublyLinkedList::setIteratorMode ( int $mode )

    mixed SplDoublyLinkedList::shift ( void )

    mixed SplDoublyLinkedList::top ( void )

    void SplDoublyLinkedList::unshift ( mixed $value )

    bool SplDoublyLinkedList::valid ( void )
}
```

SplDoublyLinkedList::bottom

SplDoublyLinkedList::bottom -- Peaks at the node from the beginning of the doubly linked list

Description

mixed **SplDoublyLinkedList::bottom** (void)

Parameters

This function has no parameters.

Return Values

The value of the first node.

SplDoublyLinkedList::__construct

SplDoublyLinkedList::__construct -- Constructs a new doubly linked list

Description

SplDoublyLinkedList::__construct (void)

This constructs a new empty doubly linked list.

Parameters

This function has no parameters.

Return Values

No value is returned.

Examples

Example #13 - SplDoublyLinkedList::__construct() **example**

```
<?php
$dll = new SplDoublyLinkedList();

$dll->push(2);
$dll->push(3);
$dll->unshift(5);

var_dump($dll);
?>
```

The above example will output:

```
object(SplDoublyLinkedList)#1 (2) {
 ["flags":"SplDoublyLinkedList":private]=>
 int(0)
 ["dllist":"SplDoublyLinkedList":private]=>
 array(3) {
   [0]=>
   int(5)
   [1]=>
   int(2)
   [2]=>
   int(3)
 }
}
```

SplDoublyLinkedList::count

SplDoublyLinkedList::count -- Counts the number of elements in the doubly linked list.

Description

int **SplDoublyLinkedList::count** (void)

Parameters

This function has no parameters.

Return Values

Returns the number of elements in the doubly linked list.

SplDoublyLinkedList::current

SplDoublyLinkedList::current -- Return current array entry

Description

mixed **SplDoublyLinkedList::current** (void)

Get the current doubly linked list node.

Parameters

This function has no parameters.

Return Values

The current node value.

SplDoublyLinkedList::getIteratorMode

SplDoublyLinkedList::getIteratorMode -- Returns the mode of iteration

Description

int **SplDoublyLinkedList::getIteratorMode** (void)

Parameters

This function has no parameters.

Return Values

Returns the different modes and flags that affect the iteration.

SplDoublyLinkedList::isEmpty

SplDoublyLinkedList::isEmpty -- Checks whether the doubly linked list is empty.

Description

bool **SplDoublyLinkedList::isEmpty** (void)

Parameters

This function has no parameters.

Return Values

Returns whether the doubly linked list is empty.

SplDoublyLinkedList::key

SplDoublyLinkedList::key -- Return current node index

Description

mixed **SplDoublyLinkedList::key** (void)

This function returns the current node index

Parameters

This function has no parameters.

Return Values

The current node index.

SplDoublyLinkedList::next

SplDoublyLinkedList::next -- Move to next entry

Description

void **SplDoublyLinkedList::next** (void)

Move the iterator to the next node.

Parameters

This function has no parameters.

Return Values

No value is returned.

SplDoublyLinkedList::offsetExists

SplDoublyLinkedList::offsetExists -- Returns whether the requested $index exists

Description

bool **SplDoublyLinkedList::offsetExists** (mixed `$index`)

Parameters

`index`
: The index being checked.

Return Values

TRUE if the requested *`index`* exists, otherwise **FALSE**

SplDoublyLinkedList::offsetGet

SplDoublyLinkedList::offsetGet -- Returns the value at the specified $index

Description

mixed **SplDoublyLinkedList::offsetGet** (mixed `$index`)

Parameters

`index`
: The index with the value.

Return Values

The value at the specified *`index`*.

SplDoublyLinkedList::offsetSet

SplDoublyLinkedList::offsetSet -- Sets the value at the specified $index to $newval

Description

void **SplDoublyLinkedList::offsetSet** (mixed `$index`, mixed `$newval`)

Sets the value at the specified *`index`* to *`newval`*.

Parameters

`index`
: The index being set.

`newval`
: The new value for the *`index`*.

Return Values

No value is returned.

SplDoublyLinkedList::offsetUnset

SplDoublyLinkedList::offsetUnset -- Unsets the value at the specified $index

Description

void **SplDoublyLinkedList::offsetUnset** (mixed `$index`)

Unsets the value at the specified index.

Parameters

`index`
: The index being unset.

Return Values

No value is returned.

SplDoublyLinkedList::pop

SplDoublyLinkedList::pop -- Pops a node from the end of the doubly linked list

Description

mixed **SplDoublyLinkedList::pop** (void)

Parameters

This function has no parameters.

Return Values

The value of the popped node.

SplDoublyLinkedList::push

SplDoublyLinkedList::push -- Pushes an element at the end of the doubly linked list

Description

void **SplDoublyLinkedList::push** (mixed `$value`)

Pushes *`value`* at the end of the doubly linked list.

Parameters

`value`
The value to push.

Return Values

No value is returned.

SplDoublyLinkedList::rewind

SplDoublyLinkedList::rewind -- Rewind iterator back to the start

Description

void **SplDoublyLinkedList::rewind** (void)

This rewinds the iterator to the beginning.

Parameters

This function has no parameters.

Return Values

No value is returned.

SplDoublyLinkedList::setIteratorMode

SplDoublyLinkedList::setIteratorMode -- Sets the mode of iteration

Description

void **SplDoublyLinkedList::setIteratorMode** (int `$mode`)

Parameters

`mode`

There are two orthogonal sets of modes that can be set:

- The direction of the iteration (either one or the other):
  - **SplDoublyLnkedList::IT_MODE_LIFO** (Stack style)
  - **SplDoublyLnkedList::IT_MODE_FIFO** (Queue style)
- The behavior of the iterator (either one or the other):
  - **SplDoublyLnkedList::IT_MODE_DELETE** (Elements are deleted by the iterator)
  - **SplDoublyLnkedList::IT_MODE_KEEP** (Elements are traversed by the iterator)

The default mode is: **SplDoublyLnkedList::IT_MODE_FIFO** | **SplDoublyLnkedList::IT_MODE_KEEP**

Return Values

No value is returned.

SplDoublyLinkedList::shift

SplDoublyLinkedList::shift -- Shifts a node from the beginning of the doubly linked list

Description

mixed **SplDoublyLinkedList::shift** (void)

Parameters

This function has no parameters.

Return Values

The value of the shifted node.

SplDoublyLinkedList::top

SplDoublyLinkedList::top -- Peaks at the node from the end of the doubly linked list

Description

mixed **SplDoublyLinkedList::top** (void)

Parameters

This function has no parameters.

Return Values

The value of the last node.

SplDoublyLinkedList::unshift

SplDoublyLinkedList::unshift -- Prepends the doubly linked list with an element

Description

void **SplDoublyLinkedList::unshift** (mixed `$value`)

Prepends *`value`* at the beginning of the doubly linked list.

Parameters

`value`
The value to unshift.

Return Values

No value is returned.

SplDoublyLinkedList::valid

SplDoublyLinkedList::valid -- Check whether the doubly linked list contains more nodes

Description

bool **SplDoublyLinkedList::valid** (void)

Checks if the doubly linked list contains any more nodes.

Parameters

This function has no parameters.

Return Values

Returns **TRUE** if the doubly linked list contains any more nodes, **FALSE** otherwise.

The SplStack class

Introduction

The SplStack class provides the main functionalities of a stack implemented using a doubly linked list.

Class synopsis

SplStack

SplStack extends SplDoublyLinkedList implements Iterator, ArrayAccess, Countable {

/* Methods */

SplStack::__construct (void)

void **SplStack::setIteratorMode** (int `$mode`)

/* Inherited methods */

mixed **SplDoublyLinkedList::bottom** (void)

int **SplDoublyLinkedList::count** (void)

mixed **SplDoublyLinkedList::current** (void)

int **SplDoublyLinkedList::getIteratorMode** (void)

bool **SplDoublyLinkedList::isEmpty** (void)

mixed **SplDoublyLinkedList::key** (void)

void **SplDoublyLinkedList::next** (void)

bool **SplDoublyLinkedList::offsetExists** (mixed `$index`)

mixed **SplDoublyLinkedList::offsetGet** (mixed `$index`)

void **SplDoublyLinkedList::offsetSet** (mixed `$index`, mixed `$newval`)

void **SplDoublyLinkedList::offsetUnset** (mixed `$index`)

```
    mixed SplDoublyLinkedList::pop ( void )

    void SplDoublyLinkedList::push ( mixed $value )

    void SplDoublyLinkedList::rewind ( void )

    void SplDoublyLinkedList::setIteratorMode ( int $mode )

    mixed SplDoublyLinkedList::shift ( void )

    mixed SplDoublyLinkedList::top ( void )

    void SplDoublyLinkedList::unshift ( mixed $value )

    bool SplDoublyLinkedList::valid ( void )
}
```

SplStack::__construct

SplStack::__construct -- Constructs a new stack implemented using a doubly linked list

Description

SplStack::__construct (void)

This constructs a new empty stack.

Note
This method automatically sets the iterator mode to SplDoublyLinkedList::IT_MODE_LIFO.

Parameters

This function has no parameters.

Return Values

No value is returned.

Examples

Example #14 - SplStack::__construct() **example**

```
<?php
$q = new SplStack();

$q[] = 1;
$q[] = 2;
$q[] = 3;

foreach ($q as $elem)  {
echo $elem."\n";
}
?>
```

The above example will output:

```
3
2
1
```

SplStack::setIteratorMode

SplStack::setIteratorMode -- Sets the mode of iteration

Description

void **SplStack::setIteratorMode** (int `$mode`)

Parameters

`mode`

There is only one iteration parameter you can modify.

- The behavior of the iterator (either one or the other):
  - SplDoublyLnkedList::IT_MODE_DELETE (Elements are deleted by the iterator)
  - SplDoublyLnkedList::IT_MODE_KEEP (Elements are traversed by the iterator)

The default mode is 0x2 : SplDoublyLnkedList::IT_MODE_LIFO | SplDoublyLnkedList::IT_MODE_KEEP

Warning
The direction of iteration can no longer be changer for SplStacks. Trying to do so will result in a RuntimeException being thrown.

Return Values

No value is returned.

The SplQueue class

Introduction

The SplQueue class provides the main functionalities of a queue implemented using a doubly linked list.

Class synopsis

SplQueue

SplQueue extends SplDoublyLinkedList implements Iterator, ArrayAccess, Countable {

/* Methods */

SplQueue::__construct (void)

mixed **SplQueue::dequeue** (void)

void **SplQueue::enqueue** (mixed `$value`)

void **SplQueue::setIteratorMode** (int `$mode`)

/* Inherited methods */

mixed **SplDoublyLinkedList::bottom** (void)

int **SplDoublyLinkedList::count** (void)

mixed **SplDoublyLinkedList::current** (void)

int **SplDoublyLinkedList::getIteratorMode** (void)

bool **SplDoublyLinkedList::isEmpty** (void)

mixed **SplDoublyLinkedList::key** (void)

void **SplDoublyLinkedList::next** (void)

bool **SplDoublyLinkedList::offsetExists** (mixed `$index`)

mixed **SplDoublyLinkedList::offsetGet** (mixed `$index`)

```
    void SplDoublyLinkedList::offsetSet ( mixed $index, mixed $newval )

    void SplDoublyLinkedList::offsetUnset ( mixed $index )

    mixed SplDoublyLinkedList::pop ( void )

    void SplDoublyLinkedList::push ( mixed $value )

    void SplDoublyLinkedList::rewind ( void )

    void SplDoublyLinkedList::setIteratorMode ( int $mode )

    mixed SplDoublyLinkedList::shift ( void )

    mixed SplDoublyLinkedList::top ( void )

    void SplDoublyLinkedList::unshift ( mixed $value )

    bool SplDoublyLinkedList::valid ( void )
}
```

SplQueue::__construct

SplQueue::__construct -- Constructs a new queue implemented using a doubly linked list

Description

SplQueue::__construct (void)

This constructs a new empty queue.

Note
This method automatically sets the iterator mode to SplDoublyLinkedList::IT_MODE_FIFO.

Parameters

This function has no parameters.

Return Values

No value is returned.

Examples

Example #15 - SplQueue::__construct() **example**

```
<?php
$q = new SplQueue();

$q[] = 1;
$q[] = 2;
$q[] = 3;

foreach ($q as $elem)  {
echo $elem."\n";
}
?>
```

The above example will output:

```
1
2
3
```

Example #16 - Efficiently handling tasks with SplQueue

```
<?php
$q = new SplQueue();
$q->setIteratorMode(SplQueue::IT_MODE_DELETE);

// ... enqueue some tasks on the queue ...

// process them
foreach ($q as $task) {
   // ... process $task ...

   // add new tasks on the queue
   $q[] = $newTask;
   // ...
}
?>
```

SplQueue::dequeue

SplQueue::dequeue -- Dequeues a node from the queue

Description

mixed **SplQueue::dequeue** (void)

Dequeues *value* from the top of of the queue.

Note
SplQueue::dequeue is an alias of **SplDoublyLinkedList::shift**.

Parameters

This function has no parameters.

Return Values

The value of the dequeued node.

SplQueue::enqueue

SplQueue::enqueue -- Adds an element to the queue.

Description

void **SplQueue::enqueue** (mixed `$value`)

Enqueues *`value`* at the end of the queue.

Note
SplQueue::enqueue is an alias of **SplDoublyLinkedList::push**.

Parameters

`value`
The value to enqueue.

Return Values

No value is returned.

SplQueue::setIteratorMode

SplQueue::setIteratorMode -- Sets the mode of iteration

Description

void **SplQueue::setIteratorMode** (int `$mode`)

Parameters

`mode`

There is only one iteration parameter you can modify.

- The behavior of the iterator (either one or the other):
  - SplDoublyLnkedList::IT_MODE_DELETE (Elements are deleted by the iterator)
  - SplDoublyLnkedList::IT_MODE_KEEP (Elements are traversed by the iterator)

The default mode is 0x0 : SplDoublyLnkedList::IT_MODE_FIFO | SplDoublyLnkedList::IT_MODE_KEEP

Warning
The direction of iteration can no longer be changer for SplQueues. Trying to do so will result in a RuntimeException being thrown.

Return Values

No value is returned.

The SplHeap class

Introduction

The SplHeap class provides the main functionalities of an Heap.

Class synopsis

SplHeap

abstract SplHeap implements Iterator, Countable {

/* Methods */

SplHeap::__construct (void)

abstract int **SplHeap::compare** (mixed `$value1`, mixed `$value2`)

int **SplHeap::count** (void)

mixed **SplHeap::current** (void)

mixed **SplHeap::extract** (void)

void **SplHeap::insert** (mixed `$value`)

bool **SplHeap::isEmpty** (void)

mixed **SplHeap::key** (void)

void **SplHeap::next** (void)

void **SplHeap::recoverFromCorruption** (void)

void **SplHeap::rewind** (void)

mixed **SplHeap::top** (void)

bool **SplHeap::valid** (void)

}

SplHeap::compare

SplHeap::compare -- Compare elements in order to place them correctly in the heap while sifting up.

Description

abstract int **SplHeap::compare** (mixed `$value1`, mixed `$value2`)

Compare *`value1`* with *`value2`*.

Warning
Throwing exceptions in **SplHeap::compare** can corrupt the Heap and place it in an blocked state. You can unblock is by calling **SplHeap::recoverFromCorruption**. However, some elements might not be placed correctly and it may hence break the heap-property.

Parameters

`value1`
: The value of the first node being compared.

`value2`
: The value of the second node being compared.

Return Values

Result of the comparison, positive integer if *`value1`* is greater than *`value2`*, 0 if they are equal, negative integer otherwise.

Note
Having multiple elements with the same value in a Heap is not recommended. They will end up in an arbitrary relative position.

SplHeap::__construct

SplHeap::__construct -- Constructs a new empty heap

Description

SplHeap::__construct (void)

This constructs a new empty heap.

Parameters

This function has no parameters.

Return Values

No value is returned.

SplHeap::count

SplHeap::count -- Counts the number of elements in the heap.

Description

int **SplHeap::count** (void)

Parameters

This function has no parameters.

Return Values

Returns the number of elements in the heap.

SplHeap::current

SplHeap::current -- Return current node pointed by the iterator

Description

mixed **SplHeap::current** (void)

Get the current datastructure node.

Parameters

This function has no parameters.

Return Values

The current node value.

SplHeap::extract

SplHeap::extract -- Extracts a node from top of the heap and sift up.

Description

mixed **SplHeap::extract** (void)

Parameters

This function has no parameters.

Return Values

The value of the extracted node.

SplHeap::insert

SplHeap::insert -- Inserts an element in the heap by sifting it up.

Description

void **SplHeap::insert** (mixed `$value`)

Insert *`value`* in the heap.

Parameters

`value`
The value to insert.

Return Values

No value is returned.

SplHeap::isEmpty

SplHeap::isEmpty -- Checks whether the heap is empty.

Description

bool **SplHeap::isEmpty** (void)

Parameters

This function has no parameters.

Return Values

Returns whether the heap is empty.

SplHeap::key

SplHeap::key -- Return current node index

Description

mixed **SplHeap::key** (void)

This function returns the current node index

Parameters

This function has no parameters.

Return Values

The current node index.

SplHeap::next

SplHeap::next -- Move to the next node

Description

void **SplHeap::next** (void)

Extracts the top node from the heap.

Parameters

This function has no parameters.

Return Values

No value is returned.

SplHeap::recoverFromCorruption

SplHeap::recoverFromCorruption -- Recover from the corrupted state and allow further actions on the heap.

Description

void **SplHeap::recoverFromCorruption** (void)

Parameters

This function has no parameters.

Return Values

No value is returned.

SplHeap::rewind

SplHeap::rewind -- Rewind iterator back to the start (no-op)

Description

void **SplHeap::rewind** (void)

This rewinds the iterator to the beginning. This is a no-op for heaps as the iterator is virtual and in fact never moves from the top of the heap.

Parameters

This function has no parameters.

Return Values

No value is returned.

SplHeap::top

SplHeap::top -- Peaks at the node from the top of the heap

Description

mixed **SplHeap::top** (void)

Parameters

This function has no parameters.

Return Values

The value of the node on the top.

SplHeap::valid

SplHeap::valid -- Check whether the heap contains more nodes

Description

bool **SplHeap::valid** (void)

Checks if the heap contains any more nodes.

Parameters

This function has no parameters.

Return Values

Returns **TRUE** if the heap contains any more nodes, **FALSE** otherwise.

The SplMaxHeap class

Introduction

The SplMaxHeap class provides the main functionalities of a heap, keeping the maximum on the top.

Class synopsis

SplMaxHeap

SplMaxHeap extends SplHeap implements Iterator, Countable {

/* Methods */

void **SplMaxHeap::compare** (mixed `$value1`, mixed `$value2`)

/* Inherited methods */

abstract int **SplHeap::compare** (mixed `$value1`, mixed `$value2`)

int **SplHeap::count** (void)

mixed **SplHeap::current** (void)

mixed **SplHeap::extract** (void)

void **SplHeap::insert** (mixed `$value`)

bool **SplHeap::isEmpty** (void)

mixed **SplHeap::key** (void)

void **SplHeap::next** (void)

void **SplHeap::recoverFromCorruption** (void)

void **SplHeap::rewind** (void)

mixed **SplHeap::top** (void)

bool **SplHeap::valid** (void)

}

SplMaxHeap::compare

SplMaxHeap::compare -- Compare elements in order to place them correctly in the heap while sifting up.

Description

void **SplMaxHeap::compare** (mixed `$value1`, mixed `$value2`)

Compare *`value1`* with *`value2`*.

Parameters

`value1`
: The value of the first node being compared.

`value2`
: The value of the second node being compared.

Return Values

Result of the comparison, positive integer if *`value1`* is greater than *`value2`*, 0 if they are equal, negative integer otherwise.

Note
Having multiple elements with the same value in a Heap is not recommended. They will end up in an arbitrary relative position.

The SplMinHeap class

Introduction

The SplMinHeap class provides the main functionalities of a heap, keeping the minimum on the top.

Class synopsis

SplMinHeap

SplMinHeap extends SplHeap implements Iterator, Countable {

/* Methods */

void **SplMinHeap::compare** (mixed `$value1`, mixed `$value2`)

/* Inherited methods */

abstract int **SplHeap::compare** (mixed `$value1`, mixed `$value2`)

int **SplHeap::count** (void)

mixed **SplHeap::current** (void)

mixed **SplHeap::extract** (void)

void **SplHeap::insert** (mixed `$value`)

bool **SplHeap::isEmpty** (void)

mixed **SplHeap::key** (void)

void **SplHeap::next** (void)

void **SplHeap::recoverFromCorruption** (void)

void **SplHeap::rewind** (void)

mixed **SplHeap::top** (void)

bool **SplHeap::valid** (void)

}

SplMinHeap::compare

SplMinHeap::compare -- Compare elements in order to place them correctly in the heap while sifting up.

Description

void **SplMinHeap::compare** (mixed `$value1`, mixed `$value2`)

Compare *`value1`* with *`value2`*.

Parameters

`value1`
: The value of the first node being compared.

`value2`
: The value of the second node being compared.

Return Values

Result of the comparison, positive integer if *`value1`* is lower than *`value2`*, 0 if they are equal, negative integer otherwise.

Note
Having multiple elements with the same value in a Heap is not recommended. They will end up in an arbitrary relative position.

The SplPriorityQueue class

Introduction

The SplPriorityQueue class provides the main functionalities of an prioritized queue, implemented using a heap.

Class synopsis

SplPriorityQueue

SplPriorityQueue implements Iterator, Countable {

/* Methods */

SplPriorityQueue::__construct (void)

void **SplPriorityQueue::compare** (mixed `$priority1`, mixed `$priority1`)

int **SplPriorityQueue::count** (void)

mixed **SplPriorityQueue::current** (void)

mixed **SplPriorityQueue::extract** (void)

void **SplPriorityQueue::insert** (mixed `$value`, mixed `$priority`)

bool **SplPriorityQueue::isEmpty** (void)

mixed **SplPriorityQueue::key** (void)

void **SplPriorityQueue::next** (void)

void **SplPriorityQueue::recoverFromCorruption** (void)

void **SplPriorityQueue::rewind** (void)

void **SplPriorityQueue::setExtractFlags** (int `$flags`)

mixed **SplPriorityQueue::top** (void)

bool **SplPriorityQueue::valid** (void)

}

SplPriorityQueue::compare

SplPriorityQueue::compare -- Compare priorities in order to place elements correctly in the heap while sifting up.

Description

void **SplPriorityQueue::compare** (mixed `$priority1`, mixed `$priority1`)

Compare *`priority1`* with *`priority2`*.

Parameters

`priority1`
: The priority of the first node being compared.

`priority2`
: The priority of the second node being compared.

Return Values

Result of the comparison, positive integer if *`priority1`* is greater than *`priority2`*, 0 if they are equal, negative integer otherwise.

Note
Multiple elements with the same priority will get dequeued in no particular order.

SplPriorityQueue::__construct

SplPriorityQueue::__construct -- Constructs a new empty queue

Description

SplPriorityQueue::__construct (void)

This constructs a new empty queue.

Parameters

This function has no parameters.

Return Values

No value is returned.

SplPriorityQueue::count

SplPriorityQueue::count -- Counts the number of elements in the queue.

Description

int **SplPriorityQueue::count** (void)

Parameters

This function has no parameters.

Return Values

Returns the number of elements in the queue.

SplPriorityQueue::current

SplPriorityQueue::current -- Return current node pointed by the iterator

Description

mixed **SplPriorityQueue::current** (void)

Get the current datastructure node.

Parameters

This function has no parameters.

Return Values

The value or priority (or both) of the current node, depending on the extract flag.

SplPriorityQueue::extract

SplPriorityQueue::extract -- Extracts a node from top of the heap and sift up.

Description

mixed **SplPriorityQueue::extract** (void)

Parameters

This function has no parameters.

Return Values

The value or priority (or both) of the extracted node, depending on the extract flag.

SplPriorityQueue::insert

SplPriorityQueue::insert -- Inserts an element in the queue by sifting it up.

Description

void **SplPriorityQueue::insert** (mixed `$value`, mixed `$priority`)

Insert *`value`* with the priority *`priority`* in the queue.

Parameters

`value`
: The value to insert.

`priority`
: The associated priority.

Return Values

No value is returned.

SplPriorityQueue::isEmpty

SplPriorityQueue::isEmpty -- Checks whether the queue is empty.

Description

bool **SplPriorityQueue::isEmpty** (void)

Parameters

This function has no parameters.

Return Values

Returns whether the queue is empty.

SplPriorityQueue::key

SplPriorityQueue::key -- Return current node index

Description

mixed **SplPriorityQueue::key** (void)

This function returns the current node index

Parameters

This function has no parameters.

Return Values

The current node index.

SplPriorityQueue::next

SplPriorityQueue::next -- Move to the next node

Description

void **SplPriorityQueue::next** (void)

Extracts the top node from the queue.

Parameters

This function has no parameters.

Return Values

No value is returned.

SplPriorityQueue::recoverFromCorruption

SplPriorityQueue::recoverFromCorruption -- Recover from the corrupted state and allow further actions on the queue.

Description

void **SplPriorityQueue::recoverFromCorruption** (void)

Parameters

This function has no parameters.

Return Values

No value is returned.

SplPriorityQueue::rewind

SplPriorityQueue::rewind -- Rewind iterator back to the start (no-op)

Description

void **SplPriorityQueue::rewind** (void)

This rewinds the iterator to the beginning. This is a no-op for heaps as the iterator is virtual and in fact never moves from the top of the heap.

Parameters

This function has no parameters.

Return Values

No value is returned.

SplPriorityQueue::setExtractFlags

SplPriorityQueue::setExtractFlags -- Sets the mode of extraction

Description

void **SplPriorityQueue::setExtractFlags** (int `$flags`)

Parameters

`flags`
: Defines what is extracted by **SplPriorityQueue::current**, **SplPriorityQueue::top** and **SplPriorityQueue::extract**.

  - **SplPriorityQueue::EXTR_DATA** (0x00000001): Extract the data
  - **SplPriorityQueue::EXTR_PRIORITY** (0x00000002): Extract the priority
  - **SplPriorityQueue::EXTR_BOTH** (0x00000003): Extract an array containing both

  The default mode is **SplPriorityQueue::EXTR_DATA**.

Return Values

No value is returned.

SplPriorityQueue::top

SplPriorityQueue::top -- Peaks at the node from the top of the queue

Description

mixed **SplPriorityQueue::top** (void)

Parameters

This function has no parameters.

Return Values

The value or priority (or both) of the top node, depending on the extract flag.

SplPriorityQueue::valid

SplPriorityQueue::valid -- Check whether the queue contains more nodes

Description

bool **SplPriorityQueue::valid** (void)

Checks if the queue contains any more nodes.

Parameters

This function has no parameters.

Return Values

Returns **TRUE** if the queue contains any more nodes, **FALSE** otherwise.