Phar

Introduction

The phar extension provides a way to put entire PHP applications into a single file called a "phar" (PHP Archive) for easy distribution and installation. In addition to providing this service, the phar extension also provides a file-format abstraction method for creating and manipulating tar and zip files through the PharData class, much as PDO provides a unified interface for accessing different databases. Unlike PDO, which cannot convert between different databases, Phar also can convert between tar, zip and phar file formats with a single line of code. see Phar::convertToExecutable() for one example.

What is phar? Phar archives are best characterized as a convenient way to group several files into a single file. As such, a phar archive provides a way to distribute a complete PHP application in a single file and run it from that file without the need to extract it to disk. Additionally, phar archives can be executed by PHP as easily as any other file, both on the commandline and from a web server. Phar is kind of like a thumb drive for PHP applications.

Phar implements this functionality through a Stream Wrapper. Normally, to use an external file within a PHP script, you would use **include()**

Example #1 - Using an external file

```
<?php
include '/path/to/external/file.php';
?>
```

PHP can be thought of as actually translating */path/to/external/file.php* into a stream wrapper as *file:///path/to/external/file.php*, and under the hood it does in fact use the plain file stream wrapper stream functions to access all local files.

To use a file named *file.php* contained with a phar archive */path/to/myphar.phar*, the syntax is very similar to the *file://* syntax above.

Example #2 - Using a file within a phar archive

```
<?php
include 'phar:///path/to/myphar.phar/file.php';
?>
```

In fact, one can treat a phar archive exactly as if it were an external disk, using any of fopen() -related functions, opendir() and mkdir() -related functions to read, change, or create new files and directories within the phar archive. This allows complete PHP applications to be distributed in a single file and run directly from that file.

The most common usage for a phar archive is to distribute a complete application in a single file. For instance, the PEAR Installer that is bundled with PHP versions is distributed as a phar archive. To use a phar archive distributed in this way, the archive can be executed on the command-line or via a web server.

Phar archives can be distributed as *tar* archives, *zip* archives, or as the custom *phar* file format designed specifically for the phar extension. Each file format has advantages and disadvantages. The tar and zip file formats can be read or extracted by any third-party tool that can read the format, but require the phar extension in order to run with PHP. The phar file format is customized and unique to the phar extension, and can only be created by the phar extension or the PEAR package » PHP_Archive, but has the advantage that applications created in this format will run even if the phar extension is not enabled.

In other words, even with the phar extension disabled, one can execute or include a phar-based archive. Accessing individual files within a phar archive is only possible with the phar extension unless the phar archive was created by PHP_Archive.

The phar extension is also capable of converting a phar archive from tar to zip or to phar file format in a single command:

Example #3 - Converting a phar archive from phar to tar file format

```
<?php
$phar = new Phar('myphar.phar');
$pgz = $phar->convertToExecutable(Phar::TAR, Phar::GZ); // makes
myphar.phar.tar.gz
?>
```

Phar can compress individual files or an entire archive using gzip compression or bzip2 compression, and can verify archive integrity automatically through the use of md5(), sha1(), **sha256()**, or **sha512()** signatures.

Lastly, the Phar extension is security-conscious, and disables write access to executable phar archives by default, and requires system-level disabling of the *phar.readonly* php.ini setting in order to create or modify phar archives. Normal tar and zip archives without an executable stub can always be created or modified using the PharData class.

If you are creating applications for distribution, you will want to read How to create Phar Archives. If you want more information on the differences between the three file formats that phar supports, you should read Phar, Tar and Zip.

If you are using phar applications, there are helpful tips in How to use Phar Archives

The word *phar* is a contraction of *PHP* and *Archive* and is based loosely on the *jar* (Java Archive) familiar to Java developers.

The implementation for Phar archives is based on the PEAR package » PHP_Archive, and the implementation details are similar, although the Phar extension is much more powerful. In addition, the Phar extension allows most PHP applications to be run unmodified while

PHP_Archive-based phar archives often require extensive modification in order to work.

Installing/Configuring

Requirements

Phar requires PHP 5.2.0 or newer. Additional features require the SPL extension in order to take advantage of iteration and array access to a Phar's file contents. The *phar* stream does not require any additional extensions to function.

You may optionally wish to enable the zlib and bzip2 extensions to take advantage of compressed phar support. In addition, to take advantage of OpenSSL signing, the OpenSSL extension must be enabled.

Note that a bug in the zlib.deflate stream filter fixed in PHP version 5.2.6 and newer may cause truncation of gzip and bzip2-compressed phar archives.

Installation

Information for installing this PECL extension may be found in the manual chapter titled Installation of PECL extensions. Additional information such as new releases, downloads, source files, maintainer information, and a CHANGELOG, can be located here: » http://pecl.php.net/package/phar.

To install on Windows, follow the instructions above, modify your php.ini and restart your web server.

```
extension_dir=c:/php5/exts/
extension=php_phar.dll
;; optional extensions
extension=php_bz2.dll
```

Runtime Configuration

The behaviour of these functions is affected by settings in *php.ini*.

Filesystem and Streams Configuration Options

Name	Default	Changeable	Changelog
phar.readonly	"1"	PHP_INI_ALL	
phar.require_hash	"1"	PHP_INI_ALL	
phar.extract_list	""	PHP_INI_ALL	Available from phar 1.1.0 to 1.2.3,

			removed in 2.0.0.
phar.cache_list	""	PHP_INI_SYSTEM	Available from phar 2.0.0.

Here's a short explanation of the configuration directives.

phar.readonly boolean
This option disables creation or modification of Phar archives using the *phar* stream or Phar object's write support. This setting should always be enabled on production machines, as the phar extension's convenient write support could allow straightforward creation of a php-based virus when coupled with other common security vulnerabilities.

Note

This setting can only be unset in php.ini due to security reasons. If *phar.readonly* is disabled in php.ini, the user may enable *phar.readonly* in a script or disable it later. If *phar.readonly* is enabled in php.ini, a script may harmlessly "re-enable" the INI variable, but may not disable it.

phar.require_hash boolean
This option will force all opened Phar archives to contain some kind of signature (currently MD5, SHA1, SHA256 and SHA512 are supported), and will refuse to process any Phar archive that does not contain a signature.

Note

This setting can only be unset in php.ini due to security reasons. If *phar.require_hash* is disabled in php.ini, the user may enable *phar.require_hash* in a script or disable it later. If *phar.require_hash* is enabled in php.ini, a script may harmlessly "re-enable" the INI variable, but may not disable it.

phar.extract_list string
This INI setting has been removed as of phar 2.0.0 Allows mappings from a full path to a phar archive or its alias to the location of its extracted files. The format of the parameter is *name=archive,name2=archive2*. This allows extraction of phar files to disk, and allows phar to act as a kind of mapper to extracted disk files. This is often done for performance reasons, or to assist with debugging a phar.

Example #4 - phar.extract_list usage example

```
in php.ini:
phar.extract_list =
archive=/full/path/to/archive/,arch2=/full/path/to/arch2
<?php
```

```
include "phar://archive/content.php";
include "phar://arch2/foo.php";
?>
```

phar.cache_list string

This INI setting was added in phar 2.0.0 Allows mapping phar archives to be pre-parsed at web server startup, providing a performance improvement that brings running files out of a phar archive very close to the speed of running those files from a traditional disk-based installation.

Example #5 - phar.cache_list usage example

```
in php.ini (windows):
phar.cache_list =C:\path\to\phar1.phar;C:\path\to\phar2.phar
in php.ini (unix):
phar.cache_list =/path/to/phar1.phar:/path/to/phar2.phar
```

Resource Types

The Phar extension provides the *phar* stream, which allows accessing files contained within a phar transparently. The file format of a Phar is described here

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.

Phar compression constants

Constant	Value	Description
Phar::NONE (integer)	0x00000000	no compression
Phar::COMPRESSED (integer)	0x0000F000	bitmask that can be used with file flags to determine if any compression is present
Phar::GZ (integer)	0x00001000	zlib (gzip) compression
Phar::BZ2 (integer)	0x00002000	bzip2 compression

Phar file format constants

Constant	Value	Description
Phar::SAME (integer)	0	retain the same file format
Phar::PHAR (integer)	1	phar file format
Phar::TAR (integer)	2	tar file format
Phar::ZIP (integer)	3	zip file format

Phar signature constants

Constant	Value	Description
Phar::MD5 (integer)	0x0001	signature with md5 hash algorithm
Phar::SHA1 (integer)	0x0002	signature with sha1 hash algorithm
Phar::SHA256 (integer)	0x0003	signature with sha256 hash algorithm (requires hash extension)

Phar::SHA512 (integer)	0x0004	signature with sha512 hash algorithm (requires hash extension)
Phar::OPENSSL (integer)	0x0010	signature with OpenSSL public/private key pair. This is a true, asymmetric key signature.

Phar webPhar mime override constants

Constant	Value	Description
Phar::PHP (integer)	1	used to instruct the mimeoverrides parameter of Phar::webPhar() that the extension should be parsed as a PHP file
Phar::PHPS (integer)	2	used to instruct the mimeoverrides parameter of Phar::webPhar() that the extension should be parsed as a PHP source file through highlight_file()

Using Phar Archives

Using Phar Archives: Introduction

Phar archives are similar in concept to Java JAR archives, but are tailored to the needs and to the flexibility of PHP applications. A Phar archive is used to distribute a complete PHP application or library in a single file. Unlike Java's implementation of JAR archives, no external tool is required to process or run a PHP Phar archive. A Phar archive application is used exactly like any other PHP application:

```
php coolapplication.phar
```

Using a Phar archive library is identical to using any other PHP library:

```
<?php
include 'coollibrary.phar';
?>
```

The *phar* stream wrapper provides the core of the phar extension, and is explained in detail here. The phar stream wrapper allows accessing the files within a phar archive using PHP's standard file functions fopen(), opendir(), and others that work on regular files. The *phar* stream wrapper supports all read/write operations on both files and directories.

```
<?php
include 'phar://coollibrary.phar/internal/file.php';
header('Content-type: image/jpeg');
// phars can be accessed by full path or by alias
echo file_get_contents('phar:///fullpath/to/coollibrary.phar/images/wow.jpg');
?>
```

The Phar class implements advanced functionality for accessing files and for creating phar archives. The Phar class is explained in detail here.

```
<?php
try {
   // open an existing phar
   $p = new Phar('coollibrary.phar', 0);
   // Phar extends SPL's DirectoryIterator class
   foreach (new RecursiveIteratorIterator($p) as $file) {
       // $file is a PharFileInfo class, and inherits from SplFileInfo
       echo $file->getFileName() . "\n";
```

```
        echo file_get_contents($file->getPathName()) . "\n"; // display contents;
    }
    if (isset($p['internal/file.php'])) {
        var_dump($p['internal/file.php']->getMetaData());
    }

    // create a new phar - phar.readonly must be 0 in php.ini
    // phar.readonly is enabled by default for security reasons.
    // On production servers, Phars need never be created,
    // only executed.
    if (Phar::canWrite()) {
        $p = new Phar('newphar.tar.phar', 0, 'newphar.tar.phar');
        // make this a tar-based phar archive, compressed with gzip compression
(.tar.gz)
        $p = $p->convertToExecutable(Phar::TAR, Phar::GZ);

        // create transaction - nothing is written to newphar.phar
        // until stopBuffering() is called, although temporary storage is needed
        $p->startBuffering();
        // add all files in /path/to/project, saving in the phar with the prefix
"project"
        $p->buildFromIterator(new RecursiveIteratorIterator(new
DirectoryIterator('/path/to/project')), '/path/to/');

        // add a new file via the array access API
        $p['file1.txt'] = 'Information';
        $fp = fopen('hugefile.dat', 'rb');
        // copy all data from the stream
        $p['data/hugefile.dat'] = $fp;

        if (Phar::canCompress(Phar::GZ)) {
            $p['data/hugefile.dat']->compress(Phar::GZ);
        }

        $p['images/wow.jpg'] = file_get_contents('images/wow.jpg');
        // any value can be saved as file-specific meta-data
        $p['images/wow.jpg']->setMetaData(array('mime-type' => 'image/jpeg'));
        $p['index.php'] = file_get_contents('index.php');
        $p->setMetaData(array('bootstrap' => 'index.php'));

        // save the phar archive to disk
        $p->stopBuffering();
    }
} catch (Exception $e) {
    echo 'Could not open Phar: ', $e;
}
?>
```

In addition, verification of phar file contents can be done using any of the supported symmetric hash algorithms (MD5, SHA1, SHA256 and SHA512 if ext/hash is enabled) and using asymmetric public/private key signing using OpenSSL (new in Phar 2.0.0). To take advantage of OpenSSL signing, you need to generate a public/private key pair, and use the private key to set the signature using Phar::setSignatureAlgorithm(). In addition, the public key as extracted using this code:

```
$public = openssl_get_publickey(file_get_contents('private.pem'));
$pkey = '';
openssl_pkey_export($public, $pkey);
```

must be saved adjacent to the phar archive it verifies. If the phar archive is saved as */path/to/my.phar*, the public key must be saved as */path/to/my.phar.pubkey*, or phar will be unable to verify the OpenSSL signature.

As of version 2.0.0, The Phar class also provides 3 static methods, Phar::webPhar(), Phar::mungServer() and Phar::interceptFileFuncs() that are crucial to packaging up PHP applications designed for usage on regular filesystems and for web-based applications. Phar::webPhar() implements a front controller that routes HTTP calls to the correct location within the phar archive. Phar::mungServer() is used to modify the values of the *$_SERVER* array to trick applications that process these values. Phar::interceptFileFuncs() instructs Phar to intercept calls to fopen(), file_get_contents(), opendir(), and all of the stat-based functions (file_exists(), is_readable() and so on) and route all relative paths to locations within the phar archive.

As an example, packaging up a release of the popular phpMyAdmin application for use as a phar archive requires only this simple script and then *phpMyAdmin.phar.tar.php* can be accessed as a regular file from your web server after modifying the user/password:

```
<?php
@unlink('phpMyAdmin.phar.tar.php');
copy('phpMyAdmin-2.11.3-english.tar.gz', 'phpMyAdmin.phar.tar.php');
$a = new Phar('phpMyAdmin.phar.tar.php');
$a->startBuffering();
$a["phpMyAdmin-2.11.3-english/config.inc.php"] = '<?php
/* Servers configuration */
$i = 0;

/* Server localhost (config:root) [1] */
$i++;
$cfg[\'Servers\'][$i][\'host\'] = \'localhost\';
$cfg[\'Servers\'][$i][\'extension\'] = \'mysqli\';
$cfg[\'Servers\'][$i][\'connect_type\'] = \'tcp\';
$cfg[\'Servers\'][$i][\'compress\'] = false;
$cfg[\'Servers\'][$i][\'auth_type\'] = \'config\';
$cfg[\'Servers\'][$i][\'user\'] = \'root\';
$cfg[\'Servers\'][$i][\'password\'] = \'\';


/* End of servers configuration */
if (strpos(PHP_OS, \'WIN\') !== false) {
   $cfg[\'UploadDir\'] = getcwd();
} else {
   $cfg[\'UploadDir\'] = \'/tmp/pharphpmyadmin\';
   @mkdir(\'/tmp/pharphpmyadmin\');
   @chmod(\'/tmp/pharphpmyadmin\', 0777);
}';
$a->setStub('<?php
Phar::interceptFileFuncs();
Phar::webPhar("phpMyAdmin.phar", "phpMyAdmin-2.11.3-english/index.php");
echo "phpMyAdmin is intended to be executed from a web browser\n";
exit -1;
__HALT_COMPILER();
');
$a->stopBuffering();
?>
```

Using Phar Archives: the phar stream wrapper

The Phar stream wrapper fully supports fopen() for read, write or append, unlink(), stat(), fstat(), fseek(), rename() and directory stream operations opendir() and as of version 2.0.0, rmdir() and mkdir().

Individual file compression and per-file metadata can also be manipulated in a Phar archive using stream contexts:

```
<?php
$context = stream_context_create(array('phar' =>
                                  array('compress' => Phar::GZ)),
                                  array('metadata' => array('user' =>
'cellog')));
file_put_contents('phar://my.phar/somefile.php', 0, $context);
?>
```

The *phar* stream wrapper does not operate on remote files, and cannot operate on remote files, and so is allowed even when the allow_url_fopen and allow_url_include INI options are disabled.

Although it is possible to create phar archives from scratch just using stream operations, it is best to use the functionality built into the Phar class. The stream wrapper is best used for read-only operations.

Using Phar Archives: the Phar and PharData class

The Phar class supports reading and manipulation of Phar archives, as well as iteration through inherited functionality of the RecursiveDirectoryIterator class. With support for the ArrayAccess interface, files inside a Phar archive can be accessed as if they were part of an associative array.

The PharData class extends the Phar, and allows creating and modifying non-executable (data) tar and zip archives even if *phar.readonly* =1 in php.ini. As such, PharData::setAlias() and PharData::setStub() are both disabled as the concept of alias and stub are unique to executable phar archives.

It is important to note that when creating a Phar archive, the full path should be passed to the Phar object constructor. Relative paths will fail to initialize.

Assuming that *$p* is a Phar object initialized as follows:

```
<?php
$p = new Phar('/path/to/myphar.phar', 0, 'myphar.phar');
?>
```

An empty Phar archive will be created at */path/to/myphar.phar*, or if */path/to/myphar.phar* already exists, it will be opened again. The literal *myphar.phar* demonstrates the concept of an alias that can be used to reference */path/to/myphar.phar* in URLs as in:

```
<?php
// these two calls to file_get_contents() are equivalent if
// /path/to/myphar.phar has an explicit alias of "myphar.phar"
// in its manifest, or if the phar was initialized with the
// previous example's Phar object setup
$f = file_get_contents('phar:///path/to/myphar.phar/whatever.txt');
$f = file_get_contents('phar://myphar.phar/whatever.txt');
?>
```

With the newly created *$p* Phar object, the following is possible:

- *$a = $p['file.php']* creates a PharFileInfo class that refers to the contents of *phar://myphar.phar/file.php*
- *$p['file.php'] = $v* creates a new file (*phar://myphar.phar/file.php*), or overwrites an existing file within *myphar.phar*. *$v* can be either a string or an open file pointer, in which case the entire contents of the file will be used to create the new file. Note that *$p->addFromString('file.php', $v)* is functionally equivalent to the above. Also possible is to add the contents of a file with *$p->addFile('/path/to/file.php', 'file.php')*. Lastly, an empty directory can be created with *$p->addEmptyDir('empty')*.
- *isset($p['file.php'])* can be used to determine whether *phar://myphar.phar/file.php* exists within *myphar.phar*.
- *unset($p['file.php'])* erases *phar://myphar.phar/file.php* from *myphar.phar*.

In addition, the Phar object is the only way to access Phar-specific metadata, through Phar::getMetaData(), and the only way to set or retrieve a Phar archive's PHP loader stub through Phar::getStub() and Phar::setStub(). Additionally, compression for the entire Phar archive at once can only be manipulated using the Phar class.

The full list of Phar object functionality is documented below.

The PharFileInfo class extends the SplFileInfo class, and adds several methods for manipulating Phar-specific details of a file contained within a Phar, such as manipulating compression and metadata.

Creating Phar Archives

Creating Phar Archives: Introduction

To be written fully in the near future. Before reading this, be sure to read How to use Phar Archives.

A great place to start is by reading about Phar::buildFromIterator(), and the specifics of the file format choices available for archives. A healthy understanding of what a stub is and does is crucial to phar archive creation, and so Phar::setStub() and Phar::createDefaultStub() are good places to start as well. If you are distributing a web-based application, it is crucial to know about Phar::webPhar() and related method Phar::mungServer(). Any application that accesses its own files should also consider using Phar::interceptFileFuncs().

What makes a phar a phar and not a tar or a zip?

Ingredients of all Phar archives, independent of file format

All Phar archives contain three to four sections:

- a stub
- a manifest describing the contents
- the file contents
- [optional] a signature for verifying Phar integrity (phar file format only)

Phar file stub

A Phar's stub is a simple PHP file. The smallest possible stub follows:

```
<?php __HALT_COMPILER();
```

A stub must contain as a minimum, the *__HALT_COMPILER();* token at its conclusion. Typically, a stub will contain loader functionality like so:

```
<?php
Phar::mapPhar();
include 'phar://myphar.phar/index.php';
__HALT_COMPILER();
```

There are no restrictions on the contents of a Phar stub, except for the requirement that it conclude with *__HALT_COMPILER();*. The closing PHP tag *?>* may be included or omitted, but there can be no more than 1 space between the*;* and the close tag*?>* or the phar extension will be unable to process the Phar archive's manifest.

In a tar or zip-based phar archive, the stub is stored in the*.phar/stub.php* file. The default stub for phar-based Phar archives contains approximately 7k of code to extract the contents of the phar and execute them. See Phar::createDefaultStub() for more detail.

The phar alias is stored in a tar or zip-based phar archive in the*.phar/alias.txt* file as plain text.

Head-to-head comparison of Phar, Tar and Zip

What are the good and the bad things about the three supported file formats in the phar

extension? This table attempts to address that question.

Feature matrix: Phar vs. Tar vs. Zip

Feature	Phar	Tar	Zip
Standard File Format	No	Yes	Yes
Can be executed without the Phar Extension [1]	Yes	No	No
Per-file compression	Yes	No	Yes
Whole-archive compression	Yes	Yes	No
Whole-archive signature validation	Yes	Yes	No
Web-specific application support	Yes	Yes	Yes
Per-file Meta-data	Yes	Yes	Yes
Whole-Archive Meta-data	Yes	Yes	Yes
Archive creation/modification [2]	Yes	Yes	Yes
Full support for all stream wrapper functions	Yes	Yes	Yes
Can be created/modified even if phar.readonly=1 [3]	No	Yes	Yes

Tip
[1] PHP can only directly access the contents of a Phar archive without the Phar extension if it is using a *stub* that extracts the contents of the phar archive. The stub created by Phar::createDefaultStub() extracts the phar archive and runs its contents from a temporary directory if no phar extension is found.

Tip
[2] All write access requires *phar.readonly* to be disabled in php.ini or on the command-line directly.

Tip
[3] Only tar and zip archives without*.phar* in their filename and without an executable stub*.phar/stub.php* can be created if phar.readonly=1.

Tar-based phars

Archives based on the tar file format follow the more modern USTAR file format. The design of the tar file header makes them more efficient to access than the zip file format, and almost as efficient as the phar file format. File names are limited to 255 bytes, including full path within the phar archive. There is no limit on the number of files within a tar-based phar archive. These archives can fully compressed in gzip or bzip2 format and still be executed by the Phar extension.

To compress an entire archive, use Phar::compress(). To decompress an entire archive, use Phar::decompress().

Zip-based phars

Archives based on the zip file format support several features built into the zip file format. Per-file and whole-archive metadata is stored in the zip file comment and zip archive comment as a serialized string. Pre-existing zip comments will be successfully read as a string. Per-file compression read/write is supported with zlib compression, and read access is supported with bzip2 compression. There is no limit on the number of files within a zip-based phar archive. Empty directories are stored in the zip archive as files with a trailing slash like *my/directory/*

Phar File Format

The phar file format is literally laid out as stub/manifest/contents/signature, and stores the crucial information of what is included in the phar archive in its *manifest*.

The Phar manifest is a highly optimized format that allows per-file specification of file compression, file permissions, and even user-defined meta-data such as file user or group. All values greater than 1 byte are stored in little-endian byte order, with the exception of the API version, which for historical reasons is stored as 3 nibbles in

big-endian order.

All unused flags are reserved for future use, and must not be used to store custom information. Use the per-file meta-data facility to store customized information about particular files.

The basic file format of a Phar archive manifest is as follows:

Global Phar manifest format

Size in bytes	Description
4 bytes	Length of manifest in bytes (1 MB limit)
4 bytes	Number of files in the Phar
2 bytes	API version of the Phar manifest (currently 1.0.0)
4 bytes	Global Phar bitmapped flags
4 bytes	Length of Phar alias
??	Phar alias (length based on previous)
4 bytes	Length of Phar metadata (*0* for none)
??	Serialized Phar Meta-data, stored in serialize() format
at least 24 * number of entries bytes	entries for each file

Global Phar bitmapped flags

Here are the bitmapped flags currently recognized by the Phar extension for the global Phar flat bitmap:

Bitmap values recognized

Value	Description
0x00010000	If set, this Phar contains a verification signature

0x00001000	If set, this Phar contains at least 1 file that is compressed with zlib compression
0x00002000	If set, this Phar contains at least 1 file that is compressed with bzip compression

Phar manifest file entry definition

Each file in the manifest contains the following information:

Phar Manifest file entry

Size in bytes	Description
4 bytes	Filename length in bytes
??	Filename (length specified in previous)
4 bytes	Un-compressed file size in bytes
4 bytes	Unix timestamp of file
4 bytes	Compressed file size in bytes
4 bytes	CRC32 checksum of un-compressed file contents
4 bytes	Bit-mapped File-specific flags
4 bytes	Serialized File Meta-data length (*0* for none)
??	Serialized File Meta-data, stored in serialize() format

Note that as of API version 1.1.1, empty directories are stored as filenames with a trailing slash like *my/directory/*

The File-specific bitmap values recognized are:

Bitmap values recognized

Value	Description
0x000001FF	These bits are reserved for defining specific file permissions of a file. Permissions are used for fstat() and can be used to recreate desired permissions upon extraction.
0x00001000	If set, this file is compressed with zlib compression
0x00002000	If set, this file is compressed with bzip compression

Phar Signature format

Phars containing a signature always have the signature appended to the end of the Phar archive after the loader, manifest, and file contents. The two signature formats supported at this time are MD5 and SHA1.

Signature format

Length in bytes	Description
16 or 20 bytes	The actual signature, 20 bytes for an SHA1 signature, 16 bytes for an MD5 signature, 32 bytes for an SHA256 signature, and 64 bytes for an SHA512 signature.
4 bytes	Signature flags. *0x0001* is used to define an MD5 signature, *0x0002* is used to define an SHA1 signature, *0x0004* is used to define an SHA256 signature, and *0x0008* is used to define an SHA512 signature. The SHA256 and SHA512 signature support was introduced with API version 1.1.0.
4 bytes	Magic *GBMB* used to define the presence of a signature.

The Phar class

Introduction

The Phar class provides a high-level interface to accessing and creating phar archives.

Class synopsis

Phar

Phar extends DirectoryIterator implements Countable, ArrayAccess {

/* Properties */

/* Methods */

bool **Phar::addEmptyDir** (string `$dirname`)

bool **Phar::addFile** (string `$file` [, string `$localname`])

bool **Phar::addFromString** (string `$localname`, string `$contents`)

string **Phar::apiVersion** (void)

array **Phar::buildFromDirectory** (string `$base_dir` [, string `$regex`])

array **Phar::buildFromIterator** (Iterator `$iter` [, string `$base_directory`])

bool **Phar::canCompress** ([int `$type`])

bool **Phar::canWrite** (void)

object **Phar::compress** (int `$compression` [, string `$extension`])

bool **Phar::compressAllFilesBZIP2** (void)

bool **Phar::compressAllFilesGZ** (void)

bool **Phar::compressFiles** (int `$compression`)

void **Phar::__construct** (string `$fname` [, int `$flags` [, string `$alias`]])

PharData **Phar::convertToData** ([int `$format` [, int `$compression` [, string `$`

`extension`]]])

Phar **Phar::convertToExecutable** ([int `$format` [, int `$compression` [, string `$extension`]]])

bool **Phar::copy** (string `$oldfile`, string `$newfile`)

int **Phar::count** (void)

string **Phar::createDefaultStub** (void)

object **Phar::decompress** ([string `$extension`])

bool **Phar::decompressFiles** (void)

int **Phar::delMetadata** (void)

int **Phar::delete** (string `$entry`)

int **Phar::extractTo** (string `$pathto` [, string|array `$files` [, bool `$overwrite`]])

int **Phar::getMetaData** (void)

bool **Phar::getModified** (void)

array **Phar::getSignature** (void)

string **Phar::getStub** (void)

array **Phar::getSupportedCompression** (void)

array **Phar::getSupportedSignatures** (void)

string **Phar::getVersion** (void)

int **Phar::hasMetadata** (void)

void **Phar::interceptFileFuncs** (void)

bool **Phar::isBuffering** (void)

mixed **Phar::isCompressed** (void)

bool **Phar::isFileFormat** (int `$format`)

bool **Phar::isValidPharFilename** (string `$filename` [, bool `$executable`])

bool **Phar::isWritable** (void)

mixed **Phar::loadPhar** (string `$filename` [, string `$alias`])

mixed **Phar::mapPhar** ([string `$alias` [, int `$dataoffset`]])

void **Phar::mount** (string `$pharpath`, string `$externalpath`)

void **Phar::mungServer** (array `$munglist`)

bool **Phar::offsetExists** (string `$offset`)

int **Phar::offsetGet** (string `$offset`)

void **Phar::offsetSet** (string `$offset`, string `$value`)

bool **Phar::offsetUnset** (string `$offset`)

bool **Phar::running** ([bool `$retphar`])

bool **Phar::setAlias** (string `$alias`)

void **Phar::setDefaultStub** ([string `$index` [, string `$webindex`]])

void **Phar::setMetadata** (mixed `$metadata`)

array **Phar::setSignatureAlgorithm** (int `$sigtype` [, string `$privatekey`])

void **Phar::setStub** (string `$stub`)

void **Phar::startBuffering** (void)

void **Phar::stopBuffering** (void)

bool **Phar::uncompressAllFiles** (void)

bool **Phar::unlinkArchive** (string `$archive`)

void **Phar::webPhar** (string `$alias`, string `$index`, string `$f404`, array `$mimetypes`, array `$rewrites`)
}

Phar::addEmptyDir

Phar::addEmptyDir -- Add an empty directory to the phar archive

Description

bool **Phar::addEmptyDir** (string `$dirname`)

Note
This method requires the *php.ini* setting *phar.readonly* to be set to *0* in order to work for Phar objects. Otherwise, a PharException will be thrown.

With this method, an empty directory is created with path *dirname*. This method is similar to ZipArchive::addEmptyDir().

Parameters

`dirname`
The name of the empty directory to create in the phar archive

Return Values

no return value, exception is thrown on failure.

Examples

Example #6 - A Phar::addEmptyDir() example

```
<?php
try {
   $a = new Phar('/path/to/phar.phar');

   $a->addEmptyDir('/full/path/to/file');
   // demonstrates how this file is stored
   $b = $a['full/path/to/file']->isDir();
} catch (Exception $e) {
   // handle errors here
}
?>
```

See Also

- PharData::addEmptyDir()
- Phar::addFile()
- Phar::addFromString()

Phar::addFile

Phar::addFile -- Add a file from the filesystem to the phar archive

Description

bool **Phar::addFile** (string `$file` [, string `$localname`])

Note
This method requires the *php.ini* setting *phar.readonly* to be set to *0* in order to work for Phar objects. Otherwise, a PharException will be thrown.

With this method, any file or URL can be added to the phar archive. If the optional second parameter *localname* is specified, the file will be stored in the archive with that name, otherwise the *file* parameter is used as the path to store within the archive. URLs must have a localname or an exception is thrown. This method is similar to ZipArchive::addFile() .

Parameters

`file`
: Full or relative path to a file on disk to be added to the phar archive.

`localname`
: Path that the file will be stored in the archive.

Return Values

no return value, exception is thrown on failure.

Examples

Example #7 - A Phar::addFile() example

```
<?php
try {
   $a = new Phar('/path/to/phar.phar');

   $a->addFile('/full/path/to/file');
   // demonstrates how this file is stored
   $b = $a['full/path/to/file']->getContent();

   $a->addFile('/full/path/to/file', 'my/file.txt');
```

```
    $c = $a['my/file.txt']->getContent();

    // demonstrate URL usage
    $a->addFile('http://www.example.com', 'example.html');
} catch (Exception $e) {
    // handle errors here
}
?>
```

See Also

- Phar::offsetSet()
- PharData::addFile()
- Phar::addFromString()
- Phar::addEmptyDir()

Phar::addFromString

Phar::addFromString -- Add a file from the filesystem to the phar archive

Description

bool **Phar::addFromString** (string `$localname`, string `$contents`)

Note
This method requires the *php.ini* setting *phar.readonly* to be set to *0* in order to work for Phar objects. Otherwise, a PharException will be thrown.

With this method, any string can be added to the phar archive. The file will be stored in the archive with *localname* as its path. This method is similar to ZipArchive::addFromString().

Parameters

`localname`
Path that the file will be stored in the archive.

`contents`
The file contents to store

Return Values

no return value, exception is thrown on failure.

Examples

Example #8 - A Phar::addFromString() example

```
<?php
try {
   $a = new Phar('/path/to/phar.phar');

   $a->addFromString('path/to/file.txt', 'my simple file');
   $b = $a['path/to/file.txt']->getContent();

   // to add contents from a stream handle for large files, use offsetSet()
   $c = fopen('/path/to/hugefile.bin');
   $a['largefile.bin'] = $c;
   fclose($c);
} catch (Exception $e) {
```

```
    // handle errors here
}
?>
```

See Also

- Phar::offsetSet()
- PharData::addFromString()
- Phar::addFile()
- Phar::addEmptyDir()

Phar::apiVersion

Phar::apiVersion -- Returns the api version

Description

string **Phar::apiVersion** (void)

Return the API version of the phar file format that will be used when creating phars. The Phar extension supports reading API version 1.0.0 or newer. API version 1.1.0 is required for SHA-256 and SHA-512 hash, and API version 1.1.1 is required to store empty directories.

Parameters

Return Values

The API version string as in *"1.0.0"*.

Examples

Example #9 - A Phar::apiVersion() example

```
<?php
echo Phar::apiVersion();
?>
```

The above example will output:

```
1.1.1
```

Phar::buildFromDirectory

Phar::buildFromDirectory -- Construct a phar archive from the files within a directory.

Description

array **Phar::buildFromDirectory** (string `$base_dir` [, string `$regex`])

Note
This method requires the *php.ini* setting *phar.readonly* to be set to *0* in order to work for Phar objects. Otherwise, a PharException will be thrown.

Populate a phar archive from directory contents. The optional second parameter is a regular expression (pcre) that is used to exclude files. Any filename that matches the regular expression will be included, all others will be excluded. For more fine-grained control, use Phar::buildFromIterator().

Parameters

`base_dir`
: The full or relative path to the directory that contains all files to add to the archive.

`regex`
: An optional pcre regular expression that is used to filter the list of files. Only file paths matching the regular expression will be included in the archive.

Return Values

Phar::buildFromDirectory() returns an associative array mapping internal path of file to the full path of the file on the filesystem.

Errors/Exceptions

This method throws BadMethodCallException when unable to instantiate the internal directory iterators, or a PharException if there were errors saving the phar archive.

Examples

Example #10 - A Phar::buildFromDirectory() example

```
<?php
// create with alias "project.phar"
$phar = new Phar('project.phar', 0, 'project.phar');
// add all files in the project
$phar->buildFromDirectory(dirname(__FILE__) . '/project');
$phar->setStub($phar->createDefaultWebStub('cli/index.php',
'www/index.php'));

$phar2 = new Phar('project2.phar', 0, 'project2.phar');
// add all files in the project, only include php files
$phar->buildFromDirectory(dirname(__FILE__) . '/project', '/\.php$/');
$phar->setStub($phar->createDefaultWebStub('cli/index.php',
'www/index.php'));
?>
```

See Also

- Phar::buildFromIterator()

Phar::buildFromIterator

Phar::buildFromIterator -- Construct a phar archive from an iterator.

Description

array **Phar::buildFromIterator** (Iterator `$iter` [, string `$base_directory`])

Note
This method requires the *php.ini* setting *phar.readonly* to be set to *0* in order to work for Phar objects. Otherwise, a PharException will be thrown.

Populate a phar archive from an iterator. Two styles of iterators are supported, iterators that map the filename within the phar to the name of a file on disk, and iterators like DirectoryIterator that return SplFileInfo objects. For iterators that return SplFileInfo objects, the second parameter is required.

Examples

Example #11 - A Phar::buildFromIterator() with SplFileInfo

For most phar archives, the archive will reflect an actual directory layout, and the second style is the most useful. For instance, to create a phar archive containing the files in this sample directory layout:

```
/path/to/project/
                config/
                       dist.xml
                       debug.xml
                lib/
                    file1.php
                    file2.php
                src/
                    processthing.php
                www/
                    index.php
                cli/
                    index.php
```

This code could be used to add these files to the "project.phar" phar archive:

```
<?php
// create with alias "project.phar"
$phar = new Phar('project.phar', 0, 'project.phar');
$phar->buildFromIterator(
```

```
   new RecursiveIteratorIterator(
    new RecursiveDirectoryIterator('/path/to/project')),
   '/path/to/project');
$phar->setStub($phar->createDefaultWebStub('cli/index.php',
'www/index.php'));
?>
```

The file project.phar can then be used immediately. **buildFromIterator()** does not set values such as compression, metadata, and this can be done after creating the phar archive.

As an interesting note, **buildFromIterator()** can also be used to copy the contents of an existing phar archive, as the Phar object descends from DirectoryIterator:

```
<?php
// create with alias "project.phar"
$phar = new Phar('project.phar', 0, 'project.phar');
$phar->buildFromIterator(
   new RecursiveIteratorIterator(
    new Phar('/path/to/anotherphar.phar')),
   'phar:///path/to/anotherphar.phar/path/to/project');
$phar->setStub($phar->createDefaultWebStub('cli/index.php',
'www/index.php'));
?>
```

Example #12 - A Phar::buildFromIterator() with other iterators

The second form of the iterator can be used with any iterator that returns a key => value mapping, such as an ArrayIterator:

```
<?php
// create with alias "project.phar"
$phar = new Phar('project.phar', 0, 'project.phar');
$phar->buildFromIterator(
   new ArrayIterator(
    array(
       'internal/file.php' => dirname(__FILE__) . '/somefile.php',
       'another/file.jpg' => fopen('/path/to/bigfile.jpg', 'rb'),
    )));
$phar->setStub($phar->createDefaultWebStub('cli/index.php',
'www/index.php'));
?>
```

Parameters

iter

Any iterator that either associatively maps phar file to location or returns SplFileInfo

objects

base_directory
: For iterators that return SplFileInfo objects, the portion of each file's full path to remove when adding to the phar archive

Return Values

buildFromIterator() returns an associative array mapping internal path of file to the full path of the file on the filesystem.

Errors/Exceptions

This method returns UnexpectedValueException when the iterator returns incorrect values, such as an integer key instead of a string, a BadMethodCallException when an SplFileInfo-based iterator is passed without a *base_directory* parameter, or a PharException if there were errors saving the phar archive.

See Also

- Phar::buildFromDirectory()

Phar::canCompress

Phar::canCompress -- Returns whether phar extension supports compression using either zlib or bzip2

Description

bool **Phar::canCompress** ([int `$type`])

This should be used to test whether compression is possible prior to loading a phar archive containing compressed files.

Parameters

`type`
: Either *Phar::GZ* or *Phar::BZ2* can be used to test whether compression is possible with a specific compression algorithm (zlib or bzip2).

Return Values

TRUE if compression/decompression is available, **FALSE** if not.

Examples

Example #13 - A Phar::canCompress() example

```
<?php
if (Phar::canCompress()) {
   echo file_get_contents('phar://compressedphar.phar/internal/file.txt');
} else {
   echo 'no compression available';
}
?>
```

See Also

- PharFileInfo::getCompressedSize()
- PharFileInfo::isCompressed()
- PharFileInfo::compress()
- PharFileInfo::decompress()
- Phar::isCompressed()

- Phar::compressFiles()
- Phar::decompressFiles()
- Phar::getSupportedCompression()
- Phar::convertToExecutable()
- Phar::convertToData()

Phar::canWrite

Phar::canWrite -- Returns whether phar extension supports writing and creating phars

Description

bool **Phar::canWrite** (void)

This static method determines whether write access has been disabled in the system php.ini via the phar.readonly ini variable.

Parameters

Return Values

TRUE if write access is enabled, **FALSE** if it is disabled.

Examples

Example #14 - A Phar::canWrite() example

```
<?php
if (Phar::canWrite()) {
   file_put_contents('phar://myphar.phar/file.txt', 'hi there');
}
?>
```

See Also

- phar.readonly
- Phar::isWritable()

Phar::compress

Phar::compress -- Compresses the entire Phar archive using Gzip or Bzip2 compression

Description

object **Phar::compress** (int `$compression` [, string `$extension`])

Note
This method requires the *php.ini* setting *phar.readonly* to be set to *0* in order to work for Phar objects. Otherwise, a PharException will be thrown.

For tar-based and phar-based phar archives, this method compresses the entire archive using gzip compression or bzip2 compression. The resulting file can be processed with the gunzip command/bunzip command, or accessed directly and transparently with the Phar extension.

For Zip-based phar archives, this method fails with an exception. The zlib extension must be enabled to compress with gzip compression, the bzip2 extension must be enabled in order to compress with bzip2 compression. As with all functionality that modifies the contents of a phar, the phar.readonly INI variable must be off in order to succeed.

In addition, this method automatically renames the archive, appending*.gz*,*.bz2* or removing the extension if passed *Phar::NONE* to remove compression. Alternatively, a file extension may be specified with the second parameter.

A Phar object is returned.

Parameters

`compression`
: Compression must be one of *Phar::GZ*, *Phar::BZ2* to add compression, or *Phar::NONE* to remove compression.

`extension`
: By default, the extension is*.phar.gz* or*.phar.bz2* for compressing phar archives, and *.phar.tar.gz* or*.phar.tar.bz2* for compressing tar archives. For decompressing, the default file extensions are*.phar* and*.phar.tar*.

Errors/Exceptions

Throws BadMethodCallException if the phar.readonly INI variable is on, the zlib extension is not available, or the bzip2 extension is not enabled.

Examples

Example #15 - A Phar::compress() example

```
<?php
$p = new Phar('/path/to/my.phar', 0, 'my.phar');
$p['myfile.txt'] = 'hi';
$p['myfile2.txt'] = 'hi';
$p1 = $p->compress(Phar::GZ); // copies to /path/to/my.phar.gz
$p2 = $p->compress(Phar::BZ2); // copies to /path/to/my.phar.bz2
$p3 = $p2->compress(Phar::NONE); // exception: /path/to/my.phar already
exists
?>
```

See Also

- PharFileInfo::getCompressedSize()
- PharFileInfo::isCompressed()
- PharFileInfo::compress()
- PharFileInfo::decompress()
- PharData::compress()
- Phar::canCompress()
- Phar::isCompressed()
- Phar::decompress()
- Phar::getSupportedCompression()
- Phar::compressFiles()
- Phar::decompressFiles()

Phar::compressAllFilesBZIP2

Phar::compressAllFilesBZIP2 -- Compresses all files in the current Phar archive using Bzip2 compression

Description

bool **Phar::compressAllFilesBZIP2** (void)

Note
This method has been removed from the phar extension as of version 2.0.0. Alternative implementations are available using Phar::compress(), Phar::decompress(), Phar::compressFiles() and Phar::decompressFiles().

Note
This method requires the *php.ini* setting *phar.readonly* to be set to *0* in order to work for Phar objects. Otherwise, a PharException will be thrown.

This method compresses all files in the Phar archive using bzip2 compression. The bzip2 extension must be enabled to take advantage of this feature. In addition, if any files are already compressed using gzip compression, the zlib extension must be enabled in order to decompress the files prior to re-compressing with bzip2 compression. As with all functionality that modifies the contents of a phar, the phar.readonly INI variable must be off in order to succeed.

Errors/Exceptions

Throws BadMethodCallException if the phar.readonly INI variable is on, the bzip2 extension is not available, or if any files are compressed using gzip compression and the zlib extension is not enabled.

Examples

Example #16 - A Phar::compressAllFilesBZIP2() example

```
<?php
$p = new Phar('/path/to/my.phar', 0, 'my.phar');
$p['myfile.txt'] = 'hi';
$p['myfile2.txt'] = 'hi';
foreach ($p as $file) {
   var_dump($file->getFileName());
   var_dump($file->isCompressed());
```

```
   var_dump($file->isCompressedBZIP2());
   var_dump($file->isCompressedGZ());
}
$p->compressAllFilesBZIP2();
foreach ($p as $file) {
   var_dump($file->getFileName());
   var_dump($file->isCompressed());
   var_dump($file->isCompressedBZIP2());
   var_dump($file->isCompressedGZ());
}
?>
```

The above example will output:

```
string(10) "myfile.txt"
bool(false)
bool(false)
bool(false)
string(11) "myfile2.txt"
bool(false)
bool(false)
bool(false)
string(10) "myfile.txt"
bool(true)
bool(true)
bool(false)
string(11) "myfile2.txt"
bool(true)
bool(true)
bool(false)
```

See Also

- PharFileInfo::getCompressedSize()
- PharFileInfo::isCompressedBZIP2()
- PharFileInfo::isCompressed()
- PharFileInfo::isCompressedGZ()
- PharFileInfo::setCompressedBZIP2()
- PharFileInfo::setUncompressed()
- PharFileInfo::setCompressedGZ()
- Phar::canCompress()
- Phar::isCompressed()
- Phar::compressAllFilesGZ()
- Phar::getSupportedCompression()
- Phar::uncompressAllFiles()

Phar::compressAllFilesGZ

Phar::compressAllFilesGZ -- Compresses all files in the current Phar archive using Gzip compression

Description

bool **Phar::compressAllFilesGZ** (void)

Note
This method has been removed from the phar extension as of version 2.0.0. Alternative implementations are available using Phar::compress(), Phar::decompress(), Phar::compressFiles() and Phar::decompressFiles().

Note
This method requires the *php.ini* setting *phar.readonly* to be set to *0* in order to work for Phar objects. Otherwise, a PharException will be thrown.

For tar-based phar archives, this method compresses the entire archive using gzip compression. The resulting file can be processed with the gunzip command, or accessed directly and transparently with the Phar extension.

For Zip-based and phar-based phar archives, this method compresses all files in the Phar archive using gzip compression. The zlib extension must be enabled to take advantage of this feature. In addition, if any files are already compressed using bzip2 compression, the bzip2 extension must be enabled in order to decompress the files prior to re-compressing with gzip compression. As with all functionality that modifies the contents of a phar, the phar.readonly INI variable must be off in order to succeed.

Errors/Exceptions

Throws BadMethodCallException if the phar.readonly INI variable is on, the zlib extension is not available, or if any files are compressed using bzip2 compression and the bzip2 extension is not enabled.

Examples

Example #17 - A Phar::compressAllFilesGZ() example

```
<?php
$p = new Phar('/path/to/my.phar', 0, 'my.phar');
```

```
$p['myfile.txt'] = 'hi';
$p['myfile2.txt'] = 'hi';
foreach ($p as $file) {
   var_dump($file->getFileName());
   var_dump($file->isCompressed());
   var_dump($file->isCompressedBZIP2());
   var_dump($file->isCompressedGZ());
}
$p->compressAllFilesGZ();
foreach ($p as $file) {
   var_dump($file->getFileName());
   var_dump($file->isCompressed());
   var_dump($file->isCompressedBZIP2());
   var_dump($file->isCompressedGZ());
}
?>
```

The above example will output:

```
string(10) "myfile.txt"
bool(false)
bool(false)
bool(false)
string(11) "myfile2.txt"
bool(false)
bool(false)
bool(false)
string(10) "myfile.txt"
bool(true)
bool(false)
bool(true)
string(11) "myfile2.txt"
bool(true)
bool(false)
bool(true)
```

See Also

- PharFileInfo::getCompressedSize()
- PharFileInfo::isCompressedBZIP2()
- PharFileInfo::isCompressed()
- PharFileInfo::isCompressedGZ()
- PharFileInfo::setCompressedBZIP2()
- PharFileInfo::setUncompressed()
- PharFileInfo::setCompressedGZ()
- Phar::canCompress()
- Phar::isCompressed()
- Phar::compressAllFilesBZIP2()
- Phar::getSupportedCompression()
- Phar::uncompressAllFiles()

Phar::compressFiles

Phar::compressFiles -- Compresses all files in the current Phar archive

Description

bool **Phar::compressFiles** (int `$compression`)

Note
This method requires the *php.ini* setting *phar.readonly* to be set to *0* in order to work for Phar objects. Otherwise, a PharException will be thrown.

For tar-based phar archives, this method throws a BadMethodCallException, as compression of individual files within a tar archive is not supported by the file format. Use Phar::compress() to compress an entire tar-based phar archive.

For Zip-based and phar-based phar archives, this method compresses all files in the Phar archive using the specified compression. The zlib or bzip2 extensions must be enabled to take advantage of this feature. In addition, if any files are already compressed using bzip2/zlib compression, the respective extension must be enabled in order to decompress the files prior to re-compressing. As with all functionality that modifies the contents of a phar, the phar.readonly INI variable must be off in order to succeed.

Parameters

`compression`
: Compression must be one of *Phar::GZ*, *Phar::BZ2* to add compression, or *Phar::NONE* to remove compression.

Errors/Exceptions

Throws BadMethodCallException if the phar.readonly INI variable is on, the zlib extension is not available, or if any files are compressed using bzip2 compression and the bzip2 extension is not enabled.

Examples

Example #18 - A Phar::compressFiles() example

```
<?php
$p = new Phar('/path/to/my.phar', 0, 'my.phar');
$p['myfile.txt'] = 'hi';
```

```
$p['myfile2.txt'] = 'hi';
foreach ($p as $file) {
   var_dump($file->getFileName());
   var_dump($file->isCompressed());
   var_dump($file->isCompressed(Phar::BZ2));
   var_dump($file->isCompressed(Phar::GZ));
}
$p->compressFiles(Phar::GZ);
foreach ($p as $file) {
   var_dump($file->getFileName());
   var_dump($file->isCompressed());
   var_dump($file->isCompressed(Phar::BZ2));
   var_dump($file->isCompressed(Phar::GZ));
}
?>
```

The above example will output:

```
string(10) "myfile.txt"
bool(false)
bool(false)
bool(false)
string(11) "myfile2.txt"
bool(false)
bool(false)
bool(false)
string(10) "myfile.txt"
int(4096)
bool(false)
bool(true)
string(11) "myfile2.txt"
int(4096)
bool(false)
bool(true)
```

See Also

- PharFileInfo::getCompressedSize()
- PharFileInfo::isCompressed()
- PharFileInfo::compress()
- PharFileInfo::decompress()
- Phar::canCompress()
- Phar::isCompressed()
- Phar::decompressFiles()
- Phar::getSupportedCompression()
- Phar::compress()
- Phar::decompress()

Phar::__construct

Phar::__construct -- Construct a Phar archive object

Description

void **Phar::__construct** (string `$fname` [, int `$flags` [, string `$alias`]])

Parameters

`fname`
: Path to an existing Phar archive or to-be-created archive

`flags`
: flags to pass to parent class RecursiveDirectoryIterator. See » SPL RecursiveDirectoryIterator docs

`alias`
: Alias with which this Phar archive should be referred to in calls to stream functionality.

Errors/Exceptions

Throws BadMethodCallException if called twice, UnexpectedValueException if the phar archive can't be opened.

Examples

Example #19 - A Phar::__construct() example

```
<?php
try {
   $p = new Phar('/path/to/my.phar', CURRENT_AS_FILEINFO | KEY_AS_FILENAME,
                 'my.phar');
} catch (UnexpectedValueException $e) {
   die('Could not open my.phar');
} catch (BadMethodCallException $e) {
   echo 'technically, this cannot happen';
}
// this works now
echo file_get_contents('phar://my.phar/example.txt');
// and works as if we had typed
echo file_get_contents('phar:///path/to/my.phar/example.txt');
?>
```

Phar::convertToData

Phar::convertToData -- Convert a phar archive to a non-executable tar or zip file

Description

PharData **Phar::convertToData** ([int `$format` [, int `$compression` [, string `$extension`]]])

This method is used to convert an executable phar archive to either a tar or zip file. To make the tar or zip non-executable, the phar stub and phar alias files are removed from the newly created archive.

If no changes are specified, this method throws a BadMethodCallException if the archive is in phar file format. For archives in tar or zip file format, this method converts the archive to a non-executable archive.

If successful, the method creates a new archive on disk and returns a PharData object. The old archive is not removed from disk, and should be done manually after the process has finished.

Parameters

`format`
: This should be one of *Phar::TAR* or *Phar::ZIP*. If set to **NULL**, the existing file format will be preserved.

`compression`
: This should be one of *Phar::NONE* for no whole-archive compression, *Phar::GZ* for zlib-based compression, and *Phar::BZ2* for bzip-based compression.

`extension`
: This parameter is used to override the default file extension for a converted archive. Note that *.phar* cannot be used anywhere in the filename for a non-executable tar or zip archive. If converting to a tar-based phar archive, the default extensions are *.tar*, *.tar.gz*, and *.tar.bz2* depending on specified compression. For zip-based archives, the default extension is *.zip*.

Return Values

The method returns a PharData object on success and throws an exception on failure.

Errors/Exceptions

This method throws BadMethodCallException when unable to compress, an unknown compression method has been specified, the requested archive is buffering with

Phar::startBuffering() and has not concluded with Phar::stopBuffering(), and a PharException if any problems are encountered during the phar creation process.

Examples

Example #20 - A Phar::convertToData() example

Using Phar::convertToData():

```
<?php
try {
   $tarphar = new Phar('myphar.phar.tar');
   // note that myphar.phar.tar is *not* unlinked
   // convert it to the non-executable tar file format
   // creates myphar.tar
   $tar = $tarphar->comvertToData();
   // convert to non-executable zip format, creates myphar.zip
   $zip = $tarphar->convertToData(Phar::ZIP);
   // create myphar.tbz
   $tgz = $tarphar->convertToData(Phar::TAR, Phar::BZ2, '.tbz');
   // creates myphar.phar.tgz
   $phar = $tarphar->convertToData(Phar::PHAR); // throws exception
} catch (Exception $e) {
   // handle the error here
}
?>
```

See Also

- Phar::convertToExecutable()
- PharData::convertToExecutable()
- PharData::convertToData()

Phar::convertToExecutable

Phar::convertToExecutable -- Convert a phar archive to another executable phar archive file format

Description

Phar **Phar::convertToExecutable** ([int `$format` [, int `$compression` [, string `$extension`]]])

Note
This method requires the *php.ini* setting *phar.readonly* to be set to *0* in order to work for Phar objects. Otherwise, a PharException will be thrown.

This method is used to convert a phar archive to another file format. For instance, it can be used to create a tar-based executable phar archive from a zip-based executable phar archive, or from an executable phar archive in the phar file format. In addition, it can be used to apply whole-archive compression to a tar or phar-based archive.

If no changes are specified, this method throws a BadMethodCallException.

If successful, the method creates a new archive on disk and returns a Phar object. The old archive is not removed from disk, and should be done manually after the process has finished.

Parameters

`format`
: This should be one of *Phar::PHAR*, *Phar::TAR*, or *Phar::ZIP*. If set to **NULL**, the existing file format will be preserved.

`compression`
: This should be one of *Phar::NONE* for no whole-archive compression, *Phar::GZ* for zlib-based compression, and *Phar::BZ2* for bzip-based compression.

`extension`
: This parameter is used to override the default file extension for a converted archive. Note that all zip- and tar-based phar archives must contain*.phar* in their file extension in order to be processed as a phar archive. If converting to a phar-based archive, the default extensions are*.phar*,*.phar.gz*, or*.phar.bz2* depending on the specified compression. For tar-based phar archives, the default extensions are*.phar.tar*, *.phar.tar.gz*, and*.phar.tar.bz2*. For zip-based phar archives, the default extension is *.phar.zip*.

Return Values

The method returns a Phar object on success and throws an exception on failure.

Errors/Exceptions

This method throws BadMethodCallException when unable to compress, an unknown compression method has been specified, the requested archive is buffering with Phar::startBuffering() and has not concluded with Phar::stopBuffering(), an UnexpectedValueException if write support is disabled, and a PharException if any problems are encountered during the phar creation process.

Examples

Example #21 - A Phar::convertToExecutable() example

Using Phar::convertToExecutable():

```
<?php
try {
   $tarphar = new Phar('myphar.phar.tar');
   // convert it to the phar file format
   // note that myphar.phar.tar is *not* unlinked
   $phar = $tarphar->convertToExecutable(Phar::PHAR); // creates myphar.phar
   $phar->setStub($phar->createDefaultStub('cli.php', 'web/index.php'));
   // creates myphar.phar.tgz
   $compressed = $phar->convertToExecutable(Phar::TAR, Phar::GZ,
'.phar.tgz');
} catch (Exception $e) {
   // handle the error here
}
?>
```

See Also

- Phar::convertToData()
- PharData::convertToExecutable()
- PharData::convertToData()

Phar::copy

Phar::copy -- Copy a file internal to the phar archive to another new file within the phar

Description

bool **Phar::copy** (string `$oldfile`, string `$newfile`)

Note
This method requires the *php.ini* setting *phar.readonly* to be set to *0* in order to work for Phar objects. Otherwise, a PharException will be thrown.

Copy a file internal to the phar archive to another new file within the phar. This is an object-oriented alternative to using copy() with the phar stream wrapper.

Parameters

`oldfile`

`newfile`

Return Values

returns **TRUE** on success, but it is safer to encase method call in a try/catch block and assume success if no exception is thrown.

Errors/Exceptions

throws UnexpectedValueException if the source file does not exist, the destination file already exists, write access is disabled, opening either file fails, reading the source file fails, or a PharException if writing the changes to the phar fails.

Examples

Example #22 - A Phar::copy() example
This example shows using Phar::copy() and the equivalent stream wrapper performance of the same thing. The primary difference between the two approaches is error handling. All Phar methods throw exceptions, whereas the stream wrapper uses

trigger_error().

```
<?php
try {
   $phar = new Phar('myphar.phar');
   $phar['a'] = 'hi';
   $phar->copy('a', 'b');
   echo $phar['b']; // outputs "hi"
} catch (Exception $e) {
   // handle error
}

// the stream wrapper equivalent of the above code.
// E_WARNINGS are triggered on error rather than exceptions.
copy('phar://myphar.phar/a', 'phar//myphar.phar/c');
echo file_get_contents('phar://myphar.phar/c'); // outputs "hi"
?>
```

Phar::count

Phar::count -- Returns the number of entries (files) in the Phar archive

Description

int **Phar::count** (void)

Parameters

Return Values

The number of files contained within this phar, or *0* (the number zero) if none.

Examples

Example #23 - A Phar::count() example

```
<?php
// make sure it doesn't exist
@unlink('brandnewphar.phar');
try {
   $p = new Phar(dirname(__FILE__) . '/brandnewphar.phar', 0,
'brandnewphar.phar');
} catch (Exception $e) {
   echo 'Could not create phar:', $e;
}
echo 'The new phar has ' . $p->count() . " entries\n";
$p['file.txt'] = 'hi';
echo 'The new phar has ' . $p->count() . " entries\n";
?>
```

The above example will output:

```
The new phar has 0 entries
The new phar has 1 entries
```

Phar::createDefaultStub

Phar::createDefaultStub -- Return the PHP loader or bootstrap stub of a Phar archive

Description

string **Phar::createDefaultStub** (void)

This method is intended for creation of phar-file format-specific stubs, and is not intended for use with tar- or zip-based phar archives.

Phar archives contain a bootstrap loader, or *stub* written in PHP that is executed when the archive is executed in PHP either via include:

```
<?php
include 'myphar.phar';
?>
```

or by simple execution:

```
php myphar.phar
```

This method provides a simple and easy method to create a stub that will run a startup file from the phar archive. In addition, different files can be specified for running the phar archive from the command line versus through a web server. The loader stub also calls Phar::interceptFileFuncs() to allow easy bundling of a PHP application that accesses the file system. If the phar extension is not present, the loader stub will extract the phar archive to a temporary directory and then operate on the files. A shutdown function erases the temporary files on exit.

Return Values

Returns a string containing the contents of a customized bootstrap loader (stub) that allows the created Phar archive to work with or without the Phar extension enabled.

Errors/Exceptions

Throws UnexpectedValueException if either parameter is longer than 400 bytes.

Examples

Example #24 - A Phar::createDefaultStub() example

```
try {
   $phar = new Phar('myphar.phar');
   $phar->setStub($phar->createDefaultStub('cli.php', 'web/index.php'));
} catch (Exception $e) {
   // handle errors
}
```

See Also

- Phar::setStub()
- Phar::getStub()

Phar::decompress

Phar::decompress -- Decompresses the entire Phar archive

Description

object **Phar::decompress** ([string `$extension`])

Note
This method requires the *php.ini* setting *phar.readonly* to be set to *0* in order to work for Phar objects. Otherwise, a PharException will be thrown.

For tar-based and phar-based phar archives, this method decompresses the entire archive.

For Zip-based phar archives, this method fails with an exception. The zlib extension must be enabled to decompress an archive compressed with with gzip compression, and the bzip2 extension must be enabled in order to decompress an archive compressed with bzip2 compression. As with all functionality that modifies the contents of a phar, the phar.readonly INI variable must be off in order to succeed.

In addition, this method automatically changes the file extension of the archive,*.phar* by default for phar archives, or*.phar.tar* for tar-based phar archives. Alternatively, a file extension may be specified with the second parameter.

A Phar object is returned.

Parameters

`extension`
: For decompressing, the default file extensions are*.phar* and*.phar.tar*. Use this parameter to specify another file extension. Be aware that all executable phar archives must contain*.phar* in their filename.

Errors/Exceptions

Throws BadMethodCallException if the phar.readonly INI variable is on, the zlib extension is not available, or the bzip2 extension is not enabled.

Examples

Example #25 - A Phar::decompress() example

```
<?php
$p = new Phar('/path/to/my.phar', 0, 'my.phar.gz');
$p['myfile.txt'] = 'hi';
$p['myfile2.txt'] = 'hi';
$p3 = $p2->decompress(); // creates /path/to/my.phar
?>
```

See Also

- PharFileInfo::getCompressedSize()
- PharFileInfo::isCompressed()
- PharFileInfo::compress()
- PharFileInfo::decompress()
- PharData::compress()
- Phar::canCompress()
- Phar::isCompressed()
- Phar::compress()
- Phar::getSupportedCompression()
- Phar::compressFiles()
- Phar::decompressFiles()

Phar::decompressFiles

Phar::decompressFiles -- Decompresses all files in the current Phar archive

Description

bool **Phar::decompressFiles** (void)

Note
This method requires the *php.ini* setting *phar.readonly* to be set to *0* in order to work for Phar objects. Otherwise, a PharException will be thrown.

For tar-based phar archives, this method throws a BadMethodCallException, as compression of individual files within a tar archive is not supported by the file format. Use Phar::compress() to compress an entire tar-based phar archive.

For Zip-based and phar-based phar archives, this method decompresses all files in the Phar archive. The zlib or bzip2 extensions must be enabled to take advantage of this feature if any files are compressed using bzip2/zlib compression. As with all functionality that modifies the contents of a phar, the phar.readonly INI variable must be off in order to succeed.

Errors/Exceptions

Throws BadMethodCallException if the phar.readonly INI variable is on, the zlib extension is not available, or if any files are compressed using bzip2 compression and the bzip2 extension is not enabled.

Examples

Example #26 - A Phar::decompressFiles() example

```
<?php
$p = new Phar('/path/to/my.phar', 0, 'my.phar');
$p['myfile.txt'] = 'hi';
$p['myfile2.txt'] = 'hi';
$p->compressFiles(Phar::GZ);
foreach ($p as $file) {
   var_dump($file->getFileName());
   var_dump($file->isCompressed());
   var_dump($file->isCompressed(Phar::BZ2));
   var_dump($file->isCompressed(Phar::GZ));
}
$p->decompressFiles();
foreach ($p as $file) {
   var_dump($file->getFileName());
```

```
    var_dump($file->isCompressed());
    var_dump($file->isCompressed(Phar::BZ2));
    var_dump($file->isCompressed(Phar::GZ));
}
?>
```

The above example will output:

```
string(10) "myfile.txt"
int(4096)
bool(false)
bool(true)
string(11) "myfile2.txt"
int(4096)
bool(false)
bool(true)
string(10) "myfile.txt"
bool(false)
bool(false)
bool(false)
string(11) "myfile2.txt"
bool(false)
bool(false)
bool(false)
```

See Also

- PharFileInfo::getCompressedSize()
- PharFileInfo::isCompressed()
- PharFileInfo::compress()
- PharFileInfo::decompress()
- Phar::canCompress()
- Phar::isCompressed()
- Phar::compressFiles()
- Phar::getSupportedCompression()
- Phar::compress()
- Phar::decompress()

Phar::delMetadata

Phar::delMetadata -- Deletes the global metadata of the phar

Description

int **Phar::delMetadata** (void)

Note
This method requires the *php.ini* setting *phar.readonly* to be set to *0* in order to work for Phar objects. Otherwise, a PharException will be thrown.

Deletes the global metadata of the phar

Parameters

Return Values

returns **TRUE** on success, but it is better to check for thrown exception, and assume success if none is thrown.

Errors/Exceptions

Throws PharException if errors occur while flushing changes to disk.

Examples

Example #27 - A Phar::delMetaData() example

```
<?php
try {
   $phar = new Phar('myphar.phar');
   var_dump($phar->getMetadata());
   $phar->setMetadata("hi there");
   var_dump($phar->getMetadata());
   $phar->delMetadata();
   var_dump($phar->getMetadata());
} catch (Exception $e) {
   // handle errors
}
?>
```

The above example will output:

```
NULL
string(8) "hi there"
NULL
```

See Also

- Phar::getMetadata()
- Phar::setMetadata()
- Phar::hasMetadata()

Phar::delete

Phar::delete -- Delete a file within a phar archive

Description

int **Phar::delete** (string `$entry`)

Note
This method requires the *php.ini* setting *phar.readonly* to be set to *0* in order to work for Phar objects. Otherwise, a PharException will be thrown.

Delete a file within an archive. This is the functional equivalent of calling unlink() on the stream wrapper equivalent, as shown in the example below.

Parameters

`entry`
Path within an archive to the file to delete.

Return Values

returns **TRUE** on success, but it is better to check for thrown exception, and assume success if none is thrown.

Errors/Exceptions

Throws PharException if errors occur while flushing changes to disk.

Examples

Example #28 - A Phar::delete() example

```
<?php
try {
   $phar = new Phar('myphar.phar');
   $phar->delete('unlink/me.php');
   // this is equivalent to:
   unlink('phar://myphar.phar/unlink/me.php');
} catch (Exception $e) {
   // handle errors
}
?>
```

See Also

- PharData::delete()
- Phar::unlinkArchive()

Phar::extractTo

Phar::extractTo -- Extract the contents of a phar archive to a directory

Description

int **Phar::extractTo** (string `$pathto` [, string|array `$files` [, bool `$overwrite`]])

Note
This method requires the *php.ini* setting *phar.readonly* to be set to *0* in order to work for Phar objects. Otherwise, a PharException will be thrown.

Extract all files within a phar archive to disk. Extracted files and directories preserve permissions as stored in the archive. The optional parameters allow optional control over which files are extracted, and whether existing files on disk can be overwritten. The second parameter *files* can be either the name of a file or directory to extract, or an array of names of files and directories to extract. By default, this method will not overwrite existing files, the third parameter can be set to true to enable overwriting of files. This method is similar to ZipArchive::extractTo().

Parameters

`pathto`
: Path within an archive to the file to delete.

`files`
: The name of a file or directory to extract, or an array of files/directories to extract

`overwrite`
: **FALSE** by default, set to **TRUE** to enable overwriting existing files

Return Values

returns **TRUE** on success, but it is better to check for thrown exception, and assume success if none is thrown.

Errors/Exceptions

Throws PharException if errors occur while flushing changes to disk.

Examples

Example #29 - A Phar::extractTo() example

```
<?php
try {
   $phar = new Phar('myphar.phar');
   $phar->extractTo('/full/path'); // extract all files
   $phar->extractTo('/another/path', 'file.txt'); // extract only file.txt
   $phar->extractTo('/this/path',
       array('file1.txt', 'file2.txt')); // extract 2 files only
   $phar->extractTo('/third/path', null, true); // extract all files, and
overwrite
} catch (Exception $e) {
   // handle errors
}
?>
```

See Also

- PharData::extractTo()

Phar::getMetaData

Phar::getMetaData -- Returns phar archive meta-data

Description

int **Phar::getMetaData** (void)

Retrieve archive meta-data. Meta-data can be any PHP variable that can be serialized.

Parameters

No parameters.

Return Values

any PHP variable that can be serialized and is stored as meta-data for the Phar archive, or **NULL** if no meta-data is stored.

Examples

Example #30 - A Phar::getMetaData() example

```
<?php
// make sure it doesn't exist
@unlink('brandnewphar.phar');
try {
   $p = new Phar(dirname(__FILE__) . '/brandnewphar.phar', 0,
'brandnewphar.phar');
   $p['file.php'] = '<?php echo "hello";';
   $p->setMetaData(array('bootstrap' => 'file.php'));
   var_dump($p->getMetaData());
} catch (Exception $e) {
   echo 'Could not modify phar:', $e;
}
?>
```

The above example will output:

```
array(1) {
 ["bootstrap"]=>
 string(8) "file.php"
}
```

See Also

- Phar::setMetadata()
- Phar::delMetadata()
- Phar::hasMetadata()

Phar::getModified

Phar::getModified -- Return whether phar was modified

Description

bool **Phar::getModified** (void)

This method can be used to determine whether a phar has either had an internal file deleted, or contents of a file changed in some way.

Parameters

No parameters.

Return Values

TRUE if the phar has been modified since opened, **FALSE** if not.

Phar::getSignature

Phar::getSignature -- Return MD5/SHA1/SHA256/SHA512 signature of a Phar archive

Description

array **Phar::getSignature** (void)

Returns the verification signature of a phar archive in a hexadecimal string.

Parameters

Return Values

Array with the opened archive's signature in "hash" key and "md5", "sha1", "sha256", or "sha512" in "hash_type". This signature is a hash calculated on the entire phar's contents, and may be used to verify the integrity of the archive. A valid signature is absolutely required of all phar-based phars if the phar.require_hash INI variable is set to true.

Phar::getStub

Phar::getStub -- Return the PHP loader or bootstrap stub of a Phar archive

Description

string **Phar::getStub** (void)

Phar archives contain a bootstrap loader, or *stub* written in PHP that is executed when the archive is executed in PHP either via include:

```
<?php
include 'myphar.phar';
?>
```

or by simple execution:

```
php myphar.phar
```

Return Values

Returns a string containing the contents of the bootstrap loader (stub) of the current Phar archive.

Errors/Exceptions

Throws RuntimeException if it is not possible to read the stub from the Phar archive.

Examples

Example #31 - A Phar::getStub() example

```
$p = new Phar('/path/to/my.phar', 0, 'my.phar');
echo $p->getStub();
echo "==NEXT==\n";
$p->setStub("<?php
function __autoload($class)
{
   include 'phar://' . str_replace('_', '/', $class);
}
Phar::mapPhar('myphar.phar');
include 'phar://myphar.phar/startup.php';
__HALT_COMPILER(); ?>");
echo $p->getStub();
```

The above example will output:

```
<?php __HALT_COMPILER(); ?>
==NEXT==
<?php
function __autoload($class)
{
```

```
   include 'phar://' . str_replace('_', '/', $class);
}
Phar::mapPhar('myphar.phar');
include 'phar://myphar.phar/startup.php';
__HALT_COMPILER(); ?>
```

See Also

- Phar::setStub()
- Phar::createDefaultStub()

Phar::getSupportedCompression

Phar::getSupportedCompression -- Return array of supported compression algorithms

Description

array **Phar::getSupportedCompression** (void)

Parameters

No parameters.

Return Values

Returns an array containing any of *Phar::GZ* or *Phar::BZ2*, depending on the availability of the zlib extension or the bz2 extension.

See Also

- PharFileInfo::getCompressedSize()
- PharFileInfo::isCompressed()
- PharFileInfo::compress()
- PharFileInfo::decompress()
- Phar::compress()
- Phar::decompress()
- Phar::canCompress()
- Phar::isCompressed()
- Phar::compressFiles()
- Phar::decompressFiles()

Phar::getSupportedSignatures

Phar::getSupportedSignatures -- Return array of supported signature types

Description

array **Phar::getSupportedSignatures** (void)

Return array of supported signature types

Parameters

No parameters.

Return Values

Returns an array containing any of "MD5", "SHA-1", "SHA-256", "SHA-512".

See Also

- Phar::getSignature()
- Phar::setSignatureAlgorithm()

Phar::getVersion

Phar::getVersion -- Return version info of Phar archive

Description

string **Phar::getVersion** (void)

Returns the API version of an opened Phar archive.

Parameters

Return Values

The opened archive's API version. This is not to be confused with the API version that the loaded phar extension will use to create new phars. Each Phar archive has the API version hard-coded into its manifest. See Phar file format documentation for more information.

See Also

- Phar::apiVersion()

Phar::hasMetaData

Phar::hasMetaData -- Returns whether phar has global meta-data

Description

int **Phar::hasMetadata** (void)

Returns whether phar has global meta-data set.

Parameters

No parameters.

Return Values

Returns **TRUE** if meta-data has been set, and **FALSE** if not.

Examples

Example #32 - A Phar::hasMetaData() example

```
<?php
try {
   $phar = new Phar('myphar.phar');
   var_dump($phar->hasMetadata());
   $phar->setMetadata(array('thing' => 'hi'));
   var_dump($phar->hasMetadata());
   $phar->delMetadata();
   var_dump($phar->hasMetadata());
} catch (Exception $e) {
   // handle error
}
?>
```

The above example will output:

```
bool(false)
bool(true)
bool(false)
```

See Also

- Phar::getMetadata()
- Phar::setMetadata()

- Phar::delMetadata()

Phar::interceptFileFuncs

Phar::interceptFileFuncs -- instructs phar to intercept fopen, file_get_contents, opendir, and all of the stat-related functions

Description

void **Phar::interceptFileFuncs** (void)

instructs phar to intercept fopen(), readfile(), file_get_contents(), opendir(), and all of the stat-related functions. If any of these functions is called from within a phar archive with a relative path, the call is modified to access a file within the phar archive. Absolute paths are assumed to be attempts to load external files from the filesystem.

This function makes it possible to run PHP applications designed to run off of a hard disk as a phar application.

Parameters

No parameters.

Return Values

Examples

Example #33 - A Phar::interceptFileFuncs() example

```
<?php
Phar::interceptFileFuncs();
include 'phar://' . __FILE__ . '/file.php';
?>
```

Assuming that this phar is at */path/to/myphar.phar* and it contains *file.php* and *file2.txt*, if *file.php* contains this code:

Example #34 - A Phar::interceptFileFuncs() example

```
<?php
echo file_get_contents('file2.txt');
?>
```

Normally PHP would search the current directory for *file2.txt*, which would translate as the directory of file.php, or the current directory of a command-line user. Phar::interceptFileFuncs() instructs PHP to consider the current directory to be *phar:///path/to/myphar.phar/* and so opens *phar:///path/to/myphar.phar/file2.txt* in the above example code.

Phar::isBuffering

Phar::isBuffering -- Used to determine whether Phar write operations are being buffered, or are flushing directly to disk

Description

bool **Phar::isBuffering** (void)

This method can be used to determine whether a Phar will save changes to disk immediately, or whether a call to Phar->stopBuffering() is needed to enable saving changes.

Phar write buffering is per-archive, buffering active for the *foo.phar* Phar archive does not affect changes to the *bar.phar* Phar archive.

Examples

Example #35 - A Phar::isBuffering() example

```
<?php
$p = new Phar(dirname(__FILE__) . '/brandnewphar.phar', 0,
'brandnewphar.phar');
$p2 = new Phar('existingphar.phar');
$p['file1.txt'] = 'hi';
var_dump($p->isBuffering());
var_dump($p2->isBuffering());
?>
=2=
<?php
$p->startBuffering();
var_dump($p->isBuffering());
var_dump($p2->isBuffering());
$p->stopBuffering();
?>
=3=
<?php
var_dump($p->isBuffering());
var_dump($p2->isBuffering());
?>
```

The above example will output:

```
bool(false)
bool(false)
=2=
bool(true)
bool(false)
=3=
```

```
bool(false)
bool(false)
```

See Also

- Phar::startBuffering()
- Phar::stopBuffering()

Phar::isCompressed

Phar::isCompressed -- Returns Phar::GZ or PHAR::BZ2 if the entire phar archive is compressed (.tar.gz/tar.bz and so on)

Description

mixed **Phar::isCompressed** (void)

Note
This method requires the *php.ini* setting *phar.readonly* to be set to *0* in order to work for Phar objects. Otherwise, a PharException will be thrown.

Returns Phar::GZ or PHAR::BZ2 if the entire phar archive is compressed (.tar.gz/tar.bz and so on). Zip-based phar archives cannot be compressed as a file, and so this method will always return **FALSE** if a zip-based phar archive is queried.

Parameters

No parameters.

Return Values

Phar::GZ, *Phar::BZ2* or **FALSE**

Examples

Example #36 - A Phar::isCompressed() example

```
<?php
try {
   $phar1 = new Phar('myphar.zip.phar');
   var_dump($phar1->isCompressed());
   $phar2 = new Phar('myuncompressed.tar.phar');
   var_dump($phar2->isCompressed());
   $phar2->compressAllFilesGZ();
   var_dump($phar2->isCompressed() == Phar::GZ);
} catch (Exception $e) {
}
?>
```

The above example will output:

```
bool(false)
bool(false)
bool(true)
```

See Also

- PharFileInfo::getCompressedSize()
- PharFileInfo::isCompressed()
- PharFileInfo::decompress()
- PharFileInfo::compress()
- Phar::decompress()
- Phar::compress()
- Phar::canCompress()
- Phar::compressFiles()
- Phar::decompressFiles()
- Phar::getSupportedCompression()

Phar::isFileFormat

Phar::isFileFormat -- Returns true if the phar archive is based on the tar/phar/zip file format depending on the parameter

Description

bool **Phar::isFileFormat** (int `$format`)

Parameters

`format`
: Either *Phar::PHAR*, *Phar::TAR*, or *Phar::ZIP* to test for the format of the archive.

Return Values

Returns **TRUE** if the phar archive matches the file format requested by the parameter

Errors/Exceptions

PharException is thrown if the parameter is an unknown file format specifier.

See Also

- Phar::convertToExecutable()
- Phar::convertToData()

Phar::isValidPharFilename

Phar::isValidPharFilename -- Returns whether the given filename is a valid phar filename

Description

bool **Phar::isValidPharFilename** (string `$filename` [, bool `$executable`])

Returns whether the given filename is a valid phar filename that will be recognized as a phar archive by the phar extension. This can be used to test a name without having to instantiate a phar archive and catch the inevitable Exception that will be thrown if an invalid name is specified.

Parameters

`filename`
: The name or full path to a phar archive not yet created

`executable`
: This parameter determines whether the filename should be treated as a phar executable archive, or a data non-executable archive and is **TRUE** by default

Return Values

Returns **TRUE** if the filename is valid, **FALSE** if not.

Phar::isWritable

Phar::isWritable -- Returns true if the phar archive can be modified

Description

bool **Phar::isWritable** (void)

This method returns **TRUE** if *phar.readonly* is *0*, and the actual phar archive on disk is not read-only.

Parameters

No parameters.

Return Values

Returns **TRUE** if the phar archive can be modified

See Also

- Phar::canWrite()
- PharData::isWritable()

Phar::loadPhar

Phar::loadPhar -- Loads any phar archive with an alias

Description

mixed **Phar::loadPhar** (string `$filename` [, string `$alias`])

This can be used to read the contents of an external Phar archive. This is most useful for assigning an alias to a phar so that subsequent references to the phar can use the shorter alias, or for loading Phar archives that only contain data and are not intended for execution/inclusion in PHP scripts.

Parameters

`filename`
: the full or relative path to the phar archive to open

`alias`
: The alias that may be used to refer to the phar archive. Note that many phar archives specify an explicit alias inside the phar archive, and a PharException will be thrown if a new alias is specified in this case.

Return Values

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

Errors/Exceptions

PharException is thrown if an alias is passed in and the phar archive already has an explicit alias

Examples

Example #37 - A Phar::loadPhar() example

Phar::loadPhar can be used anywhere to load an external Phar archive, whereas Phar::mapPhar should be used in a loader stub for a Phar.

```
<?php
try {
   Phar::loadPhar('/path/to/phar.phar', 'my.phar');
   echo file_get_contents('phar://my.phar/file.txt');
} catch (PharException $e) {
   echo $e;
```

```
}
?>
```

See Also

- Phar::mapPhar()

Phar::mapPhar

Phar::mapPhar -- Reads the currently executed file (a phar) and registers its manifest

Description

mixed **Phar::mapPhar** ([string `$alias` [, int `$dataoffset`]])

This static method can only be used inside a Phar archive's loader stub in order to initialize the phar when it is directly executed, or when it is included in another script.

Parameters

`alias`
: The alias that can be used in *phar://* URLs to refer to this archive, rather than its full path.

`dataoffset`
: Unused variable, here for compatibility with PEAR's PHP_Archive.

Return Values

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

Errors/Exceptions

PharException is thrown if not called directly within PHP execution, if no __HALT_COMPILER(); token is found in the current source file, or if the file cannot be opened for reading.

Examples

Example #38 - A Phar::mapPhar() example

mapPhar should be used only inside a phar's loader stub. Use loadPhar to load an external phar into memory.

Here is a sample Phar loader stub that uses mapPhar.

```
<?php
function __autoload($class)
{
   include 'phar://me.phar/' . str_replace('_', '/', $class) . '.php';
}
try {
```

```
   Phar::mapPhar('me.phar');
   include 'phar://me.phar/startup.php';
} catch (PharException $e) {
   echo $e->getMessage();
   die('Cannot initialize Phar');
}
__HALT_COMPILER();
```

See Also

- Phar::loadPhar()

Phar::mount

Phar::mount -- Mount an external path or file to a virtual location within the phar archive

Description

void **Phar::mount** (string `$pharpath`, string `$externalpath`)

Much like the unix file system concept of mounting external devices to paths within the directory tree, Phar::mount() allows referring to external files and directories as if they were inside of an archive. This allows powerful abstraction such as referring to external configuration files as if they were inside the archive.

Parameters

`pharpath`
: The internal path within the phar archive to use as the mounted path location. If executed within a phar archive, this may be a relative path, otherwise this must be a full *phar* URL.

`externalpath`
: A path or URL to an external file or directory to mount within the phar archive

Return Values

No return. PharException is thrown on failure.

Errors/Exceptions

Throws PharException if any problems occur mounting the path.

Examples

Example #39 - A Phar::mount() example

The following example shows accessing an external configuration file as if it were a path within a phar archive.

First, the code inside of a phar archive:

```
<?php
$configuration = simplexml_load_string(file_get_contents(
   Phar::running(false) . '/config.xml'));
?>
```

Next the external code used to mount the configuration file:

```
<?php
// first set up the association between the abstract config.xml
// and the actual one on disk
Phar::mount('phar:///path/to/archive.phar/config.xml',
'/home/example/config.xml');
// now run the application
include '/path/to/archive.phar';
?>
```

Another method is to put the mounting code inside the stub of the phar archive. Here is an example of setting up a default configuration file if no user configuration is specified:

```
<?php
// first set up the association between the abstract config.xml
// and the actual one on disk
if (defined('EXTERNAL_CONFIG')) {
   Phar::mount('config.xml', EXTERNAL_CONFIG);
   if (file_exists(__DIR__ . '/extra_config.xml')) {
       Phar::mount('extra.xml', __DIR__ . '/extra_config.xml');
   }
} else {
   Phar::mount('config.xml', 'phar://' . __FILE__ . '/default_config.xml');
   Phar::mount('extra.xml', 'phar://' . __FILE__ . '/default_extra.xml');
}
// now run the application
include 'phar://' . __FILE__ . '/index.php';
__HALT_COMPILER();
?>
```

...and the code externally to load this phar archive:

```
<?php
define('EXTERNAL_CONFIG', '/home/example/config.xml');
// now run the application
include '/path/to/archive.phar';
?>
```

Phar::mungServer

Phar::mungServer -- Defines a list of up to 4 $_SERVER variables that should be modified for execution

Description

void **Phar::mungServer** (array `$munglist`)

Phar::mungServer() should only be called within the stub of a phar archive.

Defines a list of up to 4 *$_SERVER* variables that should be modified for execution. Variables that can be modified to remove traces of phar execution are *REQUEST_URI*, *PHP_SELF*, *SCRIPT_NAME* and *SCRIPT_FILENAME*.

On its own, this method does nothing. Only when combined with Phar::webPhar() does it take effect, and only when the requested file is a PHP file to be parsed. Note that the *PATH_INFO* and *PATH_TRANSLATED* variables are always modified.

The original values of variables that are modified are stored in the SERVER array with *PHAR_* prepended, so for instance *SCRIPT_NAME* would be saved as *PHAR_SCRIPT_NAME*.

Parameters

`munglist`
an array containing as string indices any of *REQUEST_URI*, *PHP_SELF*, *SCRIPT_NAME* and *SCRIPT_FILENAME*. Other values trigger an exception, and Phar::mungServer() is case-sensitive.

Return Values

No return.

Errors/Exceptions

Throws UnexpectedValueException if any problems are found with the passed in data.

Examples

Example #40 - A Phar::mungServer() example

```
<?php
// example stub
Phar::mungServer(array('REQUEST_URI'));
```

```
Phar::webPhar();
__HALT_COMPILER();
?>
```

See Also

- Phar::webPhar()
- Phar::setStub()

Phar::offsetExists

Phar::offsetExists -- determines whether a file exists in the phar

Description

bool **Phar::offsetExists** (string `$offset`)

This is an implementation of the ArrayAccess interface allowing direct manipulation of the contents of a Phar archive using array access brackets.

offsetExists() is called whenever isset() is called.

Parameters

`offset`
The filename (relative path) to look for in a Phar.

Return Values

Returns **TRUE** if the file exists within the phar, or **FALSE** if not.

Examples

Example #41 - A Phar::offsetExists() example

```
<?php
$p = new Phar(dirname(__FILE__) . '/my.phar', 0, 'my.phar');
$p['firstfile.txt'] = 'first file';
$p['secondfile.txt'] = 'second file';
// the next set of lines call offsetExists() indirectly
var_dump(isset($p['firstfile.txt']));
var_dump(isset($p['nothere.txt']));
?>
```

The above example will output:

```
bool(true)
bool(false)
```

See Also

- Phar::offsetGet()
- Phar::offsetSet()
- Phar::offsetUnset()

Phar::offsetGet

Phar::offsetGet -- get a PharFileInfo object for a specific file

Description

int **Phar::offsetGet** (string `$offset`)

This is an implementation of the ArrayAccess interface allowing direct manipulation of the contents of a Phar archive using array access brackets. offsetGet is used for retrieving files from a Phar archive.

Parameters

`offset`
The filename (relative path) to look for in a Phar.

Return Values

A PharFileInfo object is returned that can be used to iterate over a file's contents or to retrieve information about the current file.

Errors/Exceptions

This method throws BadMethodCallException if the file does not exist in the Phar archive.

Examples

Example #42 - A Phar::offsetGet() example

As with all classes that implement the ArrayAccess interface, offsetGet is automatically called when using the [] angle bracket operator

```
<?php
$p = new Phar(dirname(__FILE__) . '/myphar.phar', 0, 'myphar.phar');
$p['exists.txt'] = "file exists\n";
try {
   // automatically calls offsetGet()
   echo $p['exists.txt'];
   echo $p['doesnotexist.txt'];
} catch (BadMethodCallException $e) {
   echo $e;
}
?>
```

The above example will output:

```
file exists
Entry doesnotexist.txt does not exist
```

See Also

- Phar::offsetExists()
- Phar::offsetSet()
- Phar::offsetUnset()

Phar::offsetSet

Phar::offsetSet -- set the contents of an internal file to those of an external file

Description

void **Phar::offsetSet** (string `$offset`, string `$value`)

Note
This method requires the *php.ini* setting *phar.readonly* to be set to *0* in order to work for Phar objects. Otherwise, a PharException will be thrown.

This is an implementation of the ArrayAccess interface allowing direct manipulation of the contents of a Phar archive using array access brackets. offsetSet is used for modifying an existing file, or adding a new file to a Phar archive.

Parameters

`offset`
The filename (relative path) to modify in a Phar.

`value`
Content of the file.

Return Values

No return values.

Errors/Exceptions

if phar.readonly is *1*, BadMethodCallException is thrown, as modifying a Phar is only allowed when phar.readonly is set to *0*. Throws PharException if there are any problems flushing changes made to the Phar archive to disk.

Examples

Example #43 - A Phar::offsetSet() example
offsetSet should not be accessed directly, but instead used via array access with the *[]* operator.

```
<?php
$p = new Phar('/path/to/my.phar', 0, 'my.phar');
try {
   // calls offsetSet
   $p['file.txt'] = 'Hi there';
} catch (Exception $e) {
   echo 'Could not modify file.txt:', $e;
}
?>
```

See Also

- Phar::offsetExists()
- Phar::offsetGet()
- Phar::offsetUnset()

Phar::offsetUnset

Phar::offsetUnset -- remove a file from a phar

Description

bool **Phar::offsetUnset** (string `$offset`)

Note
This method requires the *php.ini* setting *phar.readonly* to be set to *0* in order to work for Phar objects. Otherwise, a PharException will be thrown.

This is an implementation of the ArrayAccess interface allowing direct manipulation of the contents of a Phar archive using array access brackets. offsetUnset is used for deleting an existing file, and is called by the unset() language construct.

Parameters

`offset`
The filename (relative path) to modify in a Phar.

Return Values

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

Errors/Exceptions

if phar.readonly is *1*, BadMethodCallException is thrown, as modifying a Phar is only allowed when phar.readonly is set to *0*. Throws PharException if there are any problems flushing changes made to the Phar archive to disk.

Examples

Example #44 - A Phar::offsetUnset() example

```
<?php
$p = new Phar('/path/to/my.phar', 0, 'my.phar');
try {
   // deletes file.txt from my.phar by calling offsetUnset
   unset($p['file.txt']);
} catch (Exception $e) {
   echo 'Could not delete file.txt: ', $e;
```

```
}
?>
```

See Also

- Phar::offsetExists()
- Phar::offsetGet()
- Phar::offsetSet()
- Phar::unlinkArchive()
- Phar::delete()

Phar::running

Phar::running -- Returns the full path on disk or full phar URL to the currently executing Phar archive

Description

bool **Phar::running** ([bool `$retphar`])

Returns the full path to the running phar archive. This is intended for use much like the *__FILE__* magic constant, and only has effect inside an executing phar archive.

Inside the stub of an archive, Phar::running() returns "". Simply use **__FILE__** to access the current running phar inside a stub.

Parameters

`retphar`
: **TRUE** by default. If **TRUE**, the full path on disk to the phar archive is returned. If **FALSE**, a full phar URL is returned.

Return Values

Returns **TRUE** if the filename is valid.

Examples

Example #45 - A Phar::running() example

For the following example, assume the file is within phar archive */path/to/phar/my.phar* and the file is located at path *my/file.txt* within the phar archive.

```
<?php
$a = Phar::running(); // $a is "/path/to/my.phar"
$b = Phar::running(false); // $b is "phar:///path/to/my.phar"
?>
```

Phar::setAlias

Phar::setAlias -- Set the alias for the Phar archive

Description

bool **Phar::setAlias** (string `$alias`)

Note
This method requires the *php.ini* setting *phar.readonly* to be set to *0* in order to work for Phar objects. Otherwise, a PharException will be thrown.

Set the alias for the Phar archive, and write it as the permanent alias for this phar archive. An alias can be used internally to a phar archive to ensure that use of the *phar* stream wrapper to access internal files always works regardless of the location of the phar archive on the filesystem. Another alternative is to rely upon Phar's interception of **include()** or to use Phar::interceptFileFuncs() and use relative paths.

Parameters

`alias`
A shorthand string that this archive can be referred to in *phar* stream wrapper access.

Return Values

Errors/Exceptions

Throws UnexpectedValueException when write access is disabled, and PharException if the alias is already in use or any problems were encountered flushing changes to disk.

Examples

Example #46 - A Phar::setAlias() example

```
<?php
try {
   $phar = new Phar('myphar.phar');
   $phar->setAlias('myp.phar');
} catch (Exception $e) {
   // handle error
```

```
}
?>
```

See Also

- Phar::__construct()
- Phar::interceptFileFuncs()

Phar::setDefaultStub

Phar::setDefaultStub -- Used to set the PHP loader or bootstrap stub of a Phar archive to the default loader

Description

void **Phar::setDefaultStub** ([string `$index` [, string `$webindex`]])

Note
This method requires the *php.ini* setting *phar.readonly* to be set to *0* in order to work for Phar objects. Otherwise, a PharException will be thrown.

This method is a convenience method that combines the functionality of Phar::createDefaultStub() and Phar::setStub().

Parameters

`index`
: Relative path within the phar archive to run if accessed on the command-line

`webindex`
: Relative path within the phar archive to run if accessed through a web browser

Errors/Exceptions

UnexpectedValueException is thrown if phar.readonly is enabled in php.ini. PharException is thrown if any problems are encountered flushing changes to disk.

Examples

Example #47 - A Phar::setDefaultStub() example

```
<?php
try {
   $phar = new Phar('myphar.phar');
   $phar->setDefaultStub('cli.php', 'web/index.php');
   // this is the same as:
   // $phar->setStub($phar->createDefaultStub('cli.php', 'web/index.php'));
} catch (Exception $e) {
   // handle errors
```

```
}
?>
```

See Also

- Phar::setStub()
- Phar::createDefaultStub()

Phar::setMetadata

Phar::setMetadata -- Sets phar archive meta-data

Description

void **Phar::setMetadata** (mixed `$metadata`)

Note
This method requires the *php.ini* setting *phar.readonly* to be set to *0* in order to work for Phar objects. Otherwise, a PharException will be thrown.

Phar::setMetadata() should be used to store customized data that describes something about the phar archive as a complete entity. PharFileInfo::setMetadata() should be used for file-specific meta-data. Meta-data can slow down the performance of loading a phar archive if the data is large.

Some possible uses for meta-data include specifying which file within the archive should be used to bootstrap the archive, or the location of a file manifest like » PEAR 's package.xml file. However, any useful data that describes the phar archive may be stored.

Parameters

`metadata`
Any PHP variable containing information to store that describes the phar archive

Examples

Example #48 - A Phar::setMetadata() example

```
<?php
// make sure it doesn't exist
@unlink('brandnewphar.phar');
try {
   $p = new Phar(dirname(__FILE__) . '/brandnewphar.phar', 0,
'brandnewphar.phar');
   $p['file.php'] = '<?php echo "hello"';
   $p->setMetadata(array('bootstrap' => 'file.php'));
   var_dump($p->getMetadata());
} catch (Exception $e) {
   echo 'Could not create and/or modify phar:', $e;
```

```
}
?>
```

The above example will output:

```
array(1) {
 ["bootstrap"]=>
 string(8) "file.php"
}
```

See Also

- Phar::getMetadata()
- Phar::delMetadata()
- Phar::hasMetadata()

Phar::setSignatureAlgorithm

Phar::setSignatureAlgorithm -- set the signature algorithm for a phar and apply it.

Description

array **Phar::setSignatureAlgorithm** (int `$sigtype` [, string `$privatekey`])

Note
This method requires the *php.ini* setting *phar.readonly* to be set to *0* in order to work for Phar objects. Otherwise, a PharException will be thrown.

set the signature algorithm for a phar and apply it. The signature algorithm must be one of *Phar::MD5*, *Phar::SHA1*, *Phar::SHA256*, *Phar::SHA512*, or *Phar::OPENSSL*.

Note that all phar-based and tar-based phar archives have a signature created automatically, *SHA1* by default. data tar-based archives (archives created with the PharData class) must have their signature created and set explicitly via Phar::setSignatureAlgorithm().

Parameters

`sigtype`
: One of *Phar::MD5*, *Phar::SHA1*, *Phar::SHA256*, *Phar::SHA512*, or *Phar::OPENSSL*

`privatekey`
: The contents of an OpenSSL private key, as extracted from a certificate or OpenSSL key file:

```
$private = openssl_get_privatekey(file_get_contents('private.pem'));
$pkey = '';
openssl_pkey_export($private, $pkey);
$p->setSignatureAlgorithm(Phar::OPENSSL, $pkey);
```

See phar introduction for instructions on naming and placement of the public key file.

Return Values

Errors/Exceptions

Throws UnexpectedValueException for many errors, BadMethodCallException if called for a zip-based phar archive, and a PharException if any problems occur flushing changes to disk.

See Also

- Phar::getSupportedSignatures()
- Phar::getSignature()

Phar::setStub

Phar::setStub -- Used to set the PHP loader or bootstrap stub of a Phar archive

Description

void **Phar::setStub** (string `$stub`)

Note
This method requires the *php.ini* setting *phar.readonly* to be set to *0* in order to work for Phar objects. Otherwise, a PharException will be thrown.

This method is used to add a PHP bootstrap loader stub to a new Phar archive, or to replace the loader stub in an existing Phar archive.

The loader stub for a Phar archive is used whenever an archive is included directly as in this example:

```
<?php
include 'myphar.phar';
?>
```

The loader is not accessed when including a file through the *phar* stream wrapper like so:

```
<?php
include 'phar://myphar.phar/somefile.php';
?>
```

Parameters

`stub`
A string or an open stream handle to use as the executable stub for this phar archive.

Errors/Exceptions

UnexpectedValueException is thrown if phar.readonly is enabled in php.ini. PharException is thrown if any problems are encountered flushing changes to disk.

Examples

Example #49 - A Phar::setStub() example

```
<?php
try {
   $p = new Phar(dirname(__FILE__) . '/brandnewphar.phar', 0,
'brandnewphar.phar');
   $p['a.php'] = '<?php var_dump("Hello");';
   $p->setStub('<?php var_dump("First"); Phar::mapPhar("brandnewphar.phar");
__HALT_COMPILER(); ?>');
   include 'phar://brandnewphar.phar/a.php';
   var_dump($p->getStub());
   $p['b.php'] = '<?php var_dump("World");';
   $p->setStub('<?php var_dump("Second");
Phar::mapPhar("brandnewphar.phar"); __HALT_COMPILER(); ?>');
   include 'phar://brandnewphar.phar/b.php';
   var_dump($p->getStub());
} catch (Exception $e) {
   echo 'Write operations failed on brandnewphar.phar: ', $e;
}
?>
```

The above example will output:

```
string(5) "Hello"
string(82) "<?php var_dump("First"); Phar::mapPhar("brandnewphar.phar");
__HALT_COMPILER(); ?>"
string(5) "World"
string(83) "<?php var_dump("Second"); Phar::mapPhar("brandnewphar.phar");
__HALT_COMPILER(); ?>"
```

See Also

- Phar::getStub()
- Phar::createDefaultStub()

Phar::startBuffering

Phar::startBuffering -- Start buffering Phar write operations, do not modify the Phar object on disk

Description

void **Phar::startBuffering** (void)

Although technically unnecessary, the **startBuffering()** method can provide a significant performance boost when creating or modifying a Phar archive with a large number of files. Ordinarily, every time a file within a Phar archive is created or modified in any way, the entire Phar archive will be recreated with the changes. In this way, the archive will be up-to-date with the activity performed on it.

However, this can be unnecessary when simply creating a new Phar archive, when it would make more sense to write the entire archive out at once. Similarly, it is often necessary to make a series of changes and to ensure that they all are possible before making any changes on disk, similar to the relational database concept of transactions. the **startBuffering()** / **stopBuffering()** pair of methods is provided for this purpose.

Phar write buffering is per-archive, buffering active for the *foo.phar* Phar archive does not affect changes to the *bar.phar* Phar archive.

Examples

Example #50 - A Phar::startBuffering() example

```
<?php
// make sure it doesn't exist
@unlink('brandnewphar.phar');
try {
   $p = new Phar(dirname(__FILE__) . '/brandnewphar.phar', 0,
'brandnewphar.phar');
} catch (Exception $e) {
   echo 'Could not create phar:', $e;
}
echo 'The new phar has ' . $p->count() . " entries\n";
$p->startBuffering();
$p['file.txt'] = 'hi';
$p['file2.txt'] = 'there';
$p['file2.txt']->setCompressedGZ();
$p['file3.txt'] = 'babyface';
$p['file3.txt']->setMetaData(42);
$p->setStub("<?php
function __autoload($class)
{
   include 'phar://myphar.phar/' . str_replace('_', '/', $class) . '.php';
```

```
}
Phar::mapPhar('myphar.phar');
include 'phar://myphar.phar/startup.php';
__HALT_COMPILER();");
$p->stopBuffering();
?>
```

See Also

- Phar::stopBuffering()
- Phar::isBuffering()

Phar::stopBuffering

Phar::stopBuffering -- Stop buffering write requests to the Phar archive, and save changes to disk

Description

void **Phar::stopBuffering** (void)

stopBuffering() is used in conjunction with the **startBuffering()** method. **startBuffering()** can provide a significant performance boost when creating or modifying a Phar archive with a large number of files. Ordinarily, every time a file within a Phar archive is created or modified in any way, the entire Phar archive will be recreated with the changes. In this way, the archive will be up-to-date with the activity performed on it.

However, this can be unnecessary when simply creating a new Phar archive, when it would make more sense to write the entire archive out at once. Similarly, it is often necessary to make a series of changes and to ensure that they all are possible before making any changes on disk, similar to the relational database concept of transactions. The **startBuffering()** / **stopBuffering()** pair of methods is provided for this purpose.

Phar write buffering is per-archive, buffering active for the *foo.phar* Phar archive does not affect changes to the *bar.phar* Phar archive.

Errors/Exceptions

PharException is thrown if any problems are encountered flushing changes to disk.

Examples

Example #51 - A Phar::stopBuffering() example

```
<?php
$p = new Phar(dirname(__FILE__) . '/brandnewphar.phar', 0,
'brandnewphar.phar');
$p['file1.txt'] = 'hi';
$p->startBuffering();
var_dump($p->getStub());
$p->setStub("<?php
function __autoload(\$class)
{
   include 'phar://brandnewphar.phar/' . str_replace('_', '/', \$class) .
'.php';
}
Phar::mapPhar('brandnewphar.phar');
include 'phar://brandnewphar.phar/startup.php';
__HALT_COMPILER();");
```

```
$p->stopBuffering();
var_dump($p->getStub());
?>
```

The above example will output:

```
string(24) "<?php __HALT_COMPILER();"
string(195) "<?php
function __autoload($class)
{
   include 'phar://' . str_replace('_', '/', $class);
}
Phar::mapPhar('brandnewphar.phar');
include 'phar://brandnewphar.phar/startup.php';
__HALT_COMPILER();"
```

See Also

- Phar::startBuffering()
- Phar::isBuffering()

Phar::uncompressAllFiles

Phar::uncompressAllFiles -- Uncompresses all files in the current Phar archive

Description

bool **Phar::uncompressAllFiles** (void)

Note
This method has been removed from the phar extension as of version 2.0.0. Alternative implementations are available using Phar::compress(), Phar::decompress(), Phar::compressFiles() and Phar::decompressFiles().

Note
This method requires the *php.ini* setting *phar.readonly* to be set to *0* in order to work for Phar objects. Otherwise, a PharException will be thrown.

This method decompresses all files in the Phar archive. If any files are already compressed using gzip compression, the zlib extension must be enabled in order to decompress the files, and any files compressed using bzip2 compression require the bzip2 extension to decompress the files. As with all functionality that modifies the contents of a phar, the phar.readonly INI variable must be off in order to succeed.

Errors/Exceptions

Throws BadMethodCallException if the phar.readonly INI variable is on, the bzip2 extension is not enabled and any files are compressed using bzip2 compression, or if any files are compressed using gzip compression and the zlib extension is not enabled.

Examples

Example #52 - A Phar::uncompressAllFiles() example

```
<?php
try {
   $p = new Phar('/path/to/my.phar', 0, 'my.phar');
   $p['myfile.txt'] = 'hi';
   $p['myfile2.txt'] = 'hi';
   $p->compressAllFilesGZ();
   foreach ($p as $file) {
       var_dump($file->getFileName());
       var_dump($file->isCompressed());
```

```
        var_dump($file->isCompressedBZIP2());
        var_dump($file->isCompressedGZ());
    }
    $p->uncompressAllFiles();
    foreach ($p as $file) {
        var_dump($file->getFileName());
        var_dump($file->isCompressed());
        var_dump($file->isCompressedBZIP2());
        var_dump($file->isCompressedGZ());
    }
} catch (Exception $e) {
    echo 'Write operations failed on my.phar: ', $e;
}
?>
```

The above example will output:

```
string(10) "myfile.txt"
bool(true)
bool(false)
bool(true)
string(11) "myfile2.txt"
bool(true)
bool(false)
bool(true)
string(10) "myfile.txt"
bool(false)
bool(false)
bool(false)
string(11) "myfile2.txt"
bool(false)
bool(false)
bool(false)
```

See Also

- PharFileInfo::getCompressedSize()
- PharFileInfo::isCompressedBZIP2()
- PharFileInfo::isCompressed()
- PharFileInfo::isCompressedGZ()
- PharFileInfo::setCompressedBZIP2()
- PharFileInfo::setUncompressed()
- PharFileInfo::setCompressedGZ()
- Phar::canCompress()
- Phar::isCompressed()
- Phar::compressAllFilesBZIP2()
- Phar::compressAllFilesGZ()
- Phar::getSupportedCompression()

Phar::unlinkArchive

Phar::unlinkArchive -- Completely remove a phar archive from disk and from memory

Description

bool **Phar::unlinkArchive** (string `$archive`)

Parameters

`archive`
The path on disk to the phar archive.

Return Values

Returns **TRUE** if the archive is successfully removed from disk.

Errors/Exceptions

PharException is thrown if there are any open file pointers to the phar archive, or any existing Phar, PharData, or PharFileInfo objects referring to the phar archive.

Examples

Example #53 - A Phar::unlinkArchive() example

```
<?php
// simple usage
Phar::unlinkArchive('/path/to/my.phar');

// more common example:
$p = new Phar('my.phar');
$fp = fopen('phar://my.phar/file.txt', 'r');
// this creates 'my.phar.gz'
$gp = $p->compress(Phar::GZ);
// remove all references to the archive
unset($p);
fclose($fp);
// now remove all traces of the archive
Phar::unlinkArchive('my.phar');
?>
```

See Also

- Phar::delete()
- Phar::offsetUnset()

Phar::webPhar

Phar::webPhar -- mapPhar for web-based phars. front controller for web applications

Description

void **Phar::webPhar** (string `$alias`, string `$index`, string `$f404`, array `$mimetypes`, array `$rewrites`)

Phar::mapPhar() for web-based phars. This method parses *$_SERVER['REQUEST_URI']* and routes a request from a web browser to an internal file within the phar archive. In essence, it simulates a web server, routing requests to the correct file, echoing the correct headers and parsing PHP files as needed. This powerful method is part of what makes it easy to convert an existing PHP application into a phar archive. Combined with Phar::mungServer() and Phar::interceptFileFuncs(), any web application can be used unmodified from a phar archive.

Phar::webPhar() should only be called from the stub of a phar archive (see here for more information on what a stub is).

Parameters

`alias`
: The alias that can be used in *phar://* URLs to refer to this archive, rather than its full path.

`index`
: The location within the phar of the directory index, defaults to *index.php*.

`f404`
: The location of the script to run when a file is not found. This script should output the proper HTTP 404 headers.

`mimetypes`
: An array mapping additional file extensions to MIME type. By default, these extensions are mapped to these mime types:

```
$mimes = array(
   'phps' => 2, // pass to highlight_file()
   'c' => 'text/plain',
   'cc' => 'text/plain',
   'cpp' => 'text/plain',
   'c++' => 'text/plain',
   'dtd' => 'text/plain',
   'h' => 'text/plain',
   'log' => 'text/plain',
   'rng' => 'text/plain',
   'txt' => 'text/plain',
   'xsd' => 'text/plain',
   'php' => 1, // parse as PHP
   'inc' => 1, // parse as PHP
```

```
    'avi' => 'video/avi',
    'bmp' => 'image/bmp',
    'css' => 'text/css',
    'gif' => 'image/gif',
    'htm' => 'text/html',
    'html' => 'text/html',
    'htmls' => 'text/html',
    'ico' => 'image/x-ico',
    'jpe' => 'image/jpeg',
    'jpg' => 'image/jpeg',
    'jpeg' => 'image/jpeg',
    'js' => 'application/x-javascript',
    'midi' => 'audio/midi',
    'mid' => 'audio/midi',
    'mod' => 'audio/mod',
    'mov' => 'movie/quicktime',
    'mp3' => 'audio/mp3',
    'mpg' => 'video/mpeg',
    'mpeg' => 'video/mpeg',
    'pdf' => 'application/pdf',
    'png' => 'image/png',
    'swf' => 'application/shockwave-flash',
    'tif' => 'image/tiff',
    'tiff' => 'image/tiff',
    'wav' => 'audio/wav',
    'xbm' => 'image/xbm',
    'xml' => 'text/xml',
);
```

rewrites
: An array mapping URI to internal file, simulating mod_rewrite of apache. For example:
```
array(
    'myinfo' => 'myinfo.php'
);
```
would route calls to *http://<host>/myphar.phar/myinfo* to the file *phar:///path/to/myphar.phar/myinfo.php*, preserving GET/POST. This does not quite work like mod_rewrite in that it would not match *http://<host>/myphar.phar/myinfo/another*.

Return Values

No return values

Errors/Exceptions

Throws PharException when unable to open the internal file to output, or if called from a non-stub. If an invalid array value is passed into *mimetypes* or to *rewrites*, then UnexpectedValueException is thrown.

Examples

Example #54 - A Phar::webPhar() example

With the example below, the created phar will display *Hello World* if one browses to */myphar.phar/index.php* or to */myphar.phar*, and will display the source of *index.phps* if one browses to */myphar.phar/index.phps*.

```
<?php
// creating the phar archive:
try {
   $phar = new Phar('myphar.phar');
   $phar['index.php'] = '<?php echo "Hello World"; ?>';
   $phar['index.phps'] = '<?php echo "Hello World"; ?>';
   $phar->setStub('<?php
Phar::webPhar();
__HALT_COMPILER(); ?>');
} catch (Exception $e) {
   // handle error here
}
?>
```

See Also

- Phar::mungServer()
- Phar::interceptFileFuncs()

The PharData class

Introduction

The PharData class provides a high-level interface to accessing and creating non-executable tar and zip archives. Because these archives do not contain a stub and cannot be executed by the phar extension, it is possible to create and manipulate regular zip and tar files using the PharData class even if *phar.readonly* php.ini setting is *1*.

Class synopsis

PharData

PharData extends Phar {

/* Properties */

/* Methods */

bool **PharData::addEmptyDir** (string `$dirname`)

bool **Phar::addFile** (string `$file` [, string `$localname`])

bool **PharData::addFromString** (string `$localname`, string `$contents`)

array **Phar::buildFromDirectory** (string `$base_dir` [, string `$regex`])

array **PharData::buildFromIterator** (Iterator `$iter` [, string `$base_directory`])

object **PharData::compress** (int `$compression`, string `$extension`)

bool **PharData::compressFiles** (int `$compression`)

void **PharData::__construct** (string `$fname` [, int `$flags`])

PharData **PharData::convertToData** ([int `$format` [, int `$compression` [, string `$extension`]]])

Phar **PharData::convertToExecutable** ([int `$format` [, int `$compression` [, string `$extension`]]])

bool **PharData::copy** (string `$oldfile`, string `$newfile`)

object **PharData::decompress** ([string `$extension`])

bool **PharData::decompressFiles** (void)

int **PharData::delMetadata** (void)

int **PharData::delete** (string `$entry`)

int **PharData::extractTo** (string `$pathto` [, string|array `$files` [, bool `$overwrite`]])

bool **PharData::isWritable** (void)

void **PharData::offsetSet** (string `$offset`, string `$value`)

bool **PharData::offsetUnset** (string `$offset`)

bool **PharData::setAlias** (string `$alias`)

void **PharData::setDefaultStub** ([string `$index` [, string `$webindex`]])

void **Phar::setMetadata** (mixed `$metadata`)

array **Phar::setSignatureAlgorithm** (int `$sigtype`)

void **PharData::setStub** (string `$stub`)

/* Inherited methods */

bool **Phar::addEmptyDir** (string `$dirname`)

bool **Phar::addFile** (string `$file` [, string `$localname`])

bool **Phar::addFromString** (string `$localname`, string `$contents`)

string **Phar::apiVersion** (void)

array **Phar::buildFromDirectory** (string `$base_dir` [, string `$regex`])

array **Phar::buildFromIterator** (Iterator `$iter` [, string `$base_directory`])

bool **Phar::canCompress** ([int `$type`])

bool **Phar::canWrite** (void)

object **Phar::compress** (int `$compression` [, string `$extension`])

bool **Phar::compressAllFilesBZIP2** (void)

bool **Phar::compressAllFilesGZ** (void)

bool **Phar::compressFiles** (int `$compression`)

void **Phar::__construct** (string `$fname` [, int `$flags` [, string `$alias`]])

PharData **Phar::convertToData** ([int `$format` [, int `$compression` [, string `$extension`]]])

Phar **Phar::convertToExecutable** ([int `$format` [, int `$compression` [, string `$extension`]]])

bool **Phar::copy** (string `$oldfile`, string `$newfile`)

int **Phar::count** (void)

string **Phar::createDefaultStub** (void)

object **Phar::decompress** ([string `$extension`])

bool **Phar::decompressFiles** (void)

int **Phar::delMetadata** (void)

int **Phar::delete** (string `$entry`)

int **Phar::extractTo** (string `$pathto` [, string|array `$files` [, bool `$overwrite`]])

int **Phar::getMetaData** (void)

bool **Phar::getModified** (void)

array **Phar::getSignature** (void)

string **Phar::getStub** (void)

array **Phar::getSupportedCompression** (void)

array **Phar::getSupportedSignatures** (void)

string **Phar::getVersion** (void)

int **Phar::hasMetadata** (void)

void **Phar::interceptFileFuncs** (void)

bool **Phar::isBuffering** (void)

mixed **Phar::isCompressed** (void)

bool **Phar::isFileFormat** (int `$format`)

bool **Phar::isValidPharFilename** (string `$filename` [, bool `$executable`])

bool **Phar::isWritable** (void)

mixed **Phar::loadPhar** (string `$filename` [, string `$alias`])

mixed **Phar::mapPhar** ([string `$alias` [, int `$dataoffset`]])

void **Phar::mount** (string `$pharpath`, string `$externalpath`)

void **Phar::mungServer** (array `$munglist`)

bool **Phar::offsetExists** (string `$offset`)

int **Phar::offsetGet** (string `$offset`)

void **Phar::offsetSet** (string `$offset`, string `$value`)

bool **Phar::offsetUnset** (string `$offset`)

bool **Phar::running** ([bool `$retphar`])

bool **Phar::setAlias** (string `$alias`)

void **Phar::setDefaultStub** ([string `$index` [, string `$webindex`]])

void **Phar::setMetadata** (mixed `$metadata`)

array **Phar::setSignatureAlgorithm** (int `$sigtype` [, string `$privatekey`])

void **Phar::setStub** (string `$stub`)

void **Phar::startBuffering** (void)

void **Phar::stopBuffering** (void)

bool **Phar::uncompressAllFiles** (void)

bool **Phar::unlinkArchive** (string `$archive`)

void **Phar::webPhar** (string `$alias`, string `$index`, string `$f404`, array `$mimetypes`, array `$rewrites`)
}

PharData::addEmptyDir

PharData::addEmptyDir -- Add an empty directory to the tar/zip archive

Description

bool **PharData::addEmptyDir** (string `$dirname`)

With this method, an empty directory is created with path *dirname*. This method is similar to ZipArchive::addEmptyDir().

Parameters

`dirname`
: The name of the empty directory to create in the phar archive

Return Values

no return value, exception is thrown on failure.

Examples

Example #55 - A PharData::addEmptyDir() example

```
<?php
try {
   $a = new PharData('/path/to/my.tar');

   $a->addEmptyDir('/full/path/to/file');
   // demonstrates how this file is stored
   $b = $a['full/path/to/file']->isDir();
} catch (Exception $e) {
   // handle errors here
}
?>
```

See Also

- Phar::addEmptyDir()
- PharData::addFile()
- PharData::addFromString()

PharData::addFile

PharData::addFile -- Add a file from the filesystem to the tar/zip archive

Description

bool **Phar::addFile** (string `$file` [, string `$localname`])

With this method, any file or URL can be added to the tar/zip archive. If the optional second parameter *localname* is specified, the file will be stored in the archive with that name, otherwise the *file* parameter is used as the path to store within the archive. URLs must have a localname or an exception is thrown. This method is similar to ZipArchive::addFile().

Parameters

`file`
Full or relative path to a file on disk to be added to the phar archive.

`localname`
Path that the file will be stored in the archive.

Return Values

no return value, exception is thrown on failure.

Examples

Example #56 - A PharData::addFile() example

```
<?php
try {
   $a = new PharData('/path/to/my.tar');

   $a->addFile('/full/path/to/file');
   // demonstrates how this file is stored
   $b = $a['full/path/to/file']->getContent();

   $a->addFile('/full/path/to/file', 'my/file.txt');
   $c = $a['my/file.txt']->getContent();

   // demonstrate URL usage
   $a->addFile('http://www.example.com', 'example.html');
} catch (Exception $e) {
   // handle errors here
}
?>
```

See Also

- PharData::offsetSet()
- Phar::addFile()
- PharData::addFromString()
- PharData::addEmptyDir()

PharData::addFromString

PharData::addFromString -- Add a file from the filesystem to the tar/zip archive

Description

bool **PharData::addFromString** (string `$localname`, string `$contents`)

With this method, any string can be added to the tar/zip archive. The file will be stored in the archive with *localname* as its path. This method is similar to ZipArchive::addFromString().

Parameters

`localname`
Path that the file will be stored in the archive.

`contents`
The file contents to store

Return Values

no return value, exception is thrown on failure.

Examples

Example #57 - A PharData::addFromString() example

```
<?php
try {
   $a = new PharData('/path/to/my.tar');

   $a->addFromString('path/to/file.txt', 'my simple file');
   $b = $a['path/to/file.txt']->getContent();

   // to add contents from a stream handle for large files, use offsetSet()
   $c = fopen('/path/to/hugefile.bin');
   $a['largefile.bin'] = $c;
   fclose($c);
} catch (Exception $e) {
   // handle errors here
}
?>
```

See Also

- PharData::offsetSet()
- Phar::addFromString()
- PharData::addFile()
- PharData::addEmptyDir()

PharData::buildFromDirectory

PharData::buildFromDirectory -- Construct a tar/zip archive from the files within a directory.

Description

array **Phar::buildFromDirectory** (string `$base_dir` [, string `$regex`])

Populate a tar/zip archive from directory contents. The optional second parameter is a regular expression (pcre) that is used to exclude files. Any filename that matches the regular expression will be included, all others will be excluded. For more fine-grained control, use PharData::buildFromIterator().

Parameters

`base_dir`
: The full or relative path to the directory that contains all files to add to the archive.

`regex`
: An optional pcre regular expression that is used to filter the list of files. Only file paths matching the regular expression will be included in the archive.

Return Values

Phar::buildFromDirectory() returns an associative array mapping internal path of file to the full path of the file on the filesystem.

Errors/Exceptions

This method throws BadMethodCallException when unable to instantiate the internal directory iterators, or a PharException if there were errors saving the phar archive.

Examples

Example #58 - A PharData::buildFromDirectory() example

```
<?php
$phar = new PharData('project.tar');
// add all files in the project
$phar->buildFromDirectory(dirname(__FILE__) . '/project');

$phar2 = new PharData('project2.zip');
// add all files in the project, only include php files
```

```
$phar->buildFromDirectory(dirname(__FILE__) . '/project', '/\.php$/');
?>
```

See Also

- Phar::buildFromDirectory()
- PharData::buildFromIterator()

PharData::buildFromIterator

PharData::buildFromIterator -- Construct a tar or zip archive from an iterator.

Description

array **PharData::buildFromIterator** (Iterator `$iter` [, string `$base_directory`])

Populate a tar or zip archive from an iterator. Two styles of iterators are supported, iterators that map the filename within the tar/zip to the name of a file on disk, and iterators like DirectoryIterator that return SplFileInfo objects. For iterators that return SplFileInfo objects, the second parameter is required.

Examples

Example #59 - A PharData::buildFromIterator() with SplFileInfo

For most tar/zip archives, the archive will reflect an actual directory layout, and the second style is the most useful. For instance, to create a tar/zip archive containing the files in this sample directory layout:

```
/path/to/project/
                config/
                       dist.xml
                       debug.xml
                lib/
                    file1.php
                    file2.php
                src/
                    processthing.php
                www/
                    index.php
                cli/
                    index.php
```

This code could be used to add these files to the "project.tar" tar archive:

```
<?php
$phar = new PharData('project.tar');
$phar->buildFromIterator(
   new RecursiveIteratorIterator(
    new RecursiveDirectoryIterator('/path/to/project')),
   '/path/to/project');
?>
```

The file *project.tar* can then be used immediately. **buildFromIterator()** does not set values such as compression, metadata, and this can be done after creating the tar/zip

archive.

As an interesting note, **buildFromIterator()** can also be used to copy the contents of an existing phar, tar or zip archive, as the PharData object descends from DirectoryIterator:

```
<?php
$phar = new PharData('project.tar');
$phar->buildFromIterator(
   new RecursiveIteratorIterator(
    new Phar('/path/to/anotherphar.phar')),
   'phar:///path/to/anotherphar.phar/path/to/project');
$phar->setStub($phar->createDefaultWebStub('cli/index.php',
'www/index.php'));
?>
```

Example #60 - A PharData::buildFromIterator() with other iterators

The second form of the iterator can be used with any iterator that returns a key => value mapping, such as an ArrayIterator:

```
<?php
$phar = new PharData('project.tar');
$phar->buildFromIterator(
   new ArrayIterator(
    array(
       'internal/file.php' => dirname(__FILE__) . '/somefile.php',
       'another/file.jpg' => fopen('/path/to/bigfile.jpg', 'rb'),
    )));
?>
```

Parameters

iter
: Any iterator that either associatively maps tar/zip file to location or returns SplFileInfo objects

base_directory
: For iterators that return SplFileInfo objects, the portion of each file's full path to remove when adding to the tar/zip archive

Return Values

buildFromIterator() returns an associative array mapping internal path of file to the full path of the file on the filesystem.

Errors/Exceptions

This method returns UnexpectedValueException when the iterator returns incorrect values, such as an integer key instead of a string, a BadMethodCallException when an SplFileInfo-based iterator is passed without a `base_directory` parameter, or a PharException if there were errors saving the phar archive.

See Also

- Phar::buildFromIterator()

PharData::compress

PharData::compress -- Compresses the entire tar/zip archive using Gzip or Bzip2 compression

Description

object **PharData::compress** (int `$compression`, string `$extension`)

For tar archives, this method compresses the entire archive using gzip compression or bzip2 compression. The resulting file can be processed with the gunzip command/bunzip command, or accessed directly and transparently with the Phar extension.

For zip archives, this method fails with an exception. The zlib extension must be enabled to compress with gzip compression, the bzip2 extension must be enabled in order to compress with bzip2 compression.

In addition, this method automatically renames the archive, appending*.gz*,*.bz2* or removing the extension if passed *Phar::NONE* to remove compression. Alternatively, a file extension may be specified with the second parameter.

A PharData object is returned.

Parameters

`compression`
: Compression must be one of *Phar::GZ*, *Phar::BZ2* to add compression, or *Phar::NONE* to remove compression.

`extension`
: By default, the extension is*.tar.gz* or*.tar.bz2* for compressing a tar, and*.tar* for decompressing.

Errors/Exceptions

Throws BadMethodCallException if the zlib extension is not available, or the bzip2 extension is not enabled.

Examples

Example #61 - A PharData::compress() example

```
<?php
$p = new PharData('/path/to/my.tar');
$p['myfile.txt'] = 'hi';
```

```
$p['myfile2.txt'] = 'hi';
$p1 = $p->compress(Phar::GZ); // copies to /path/to/my.phar.gz
$p2 = $p->compress(Phar::BZ2); // copies to /path/to/my.phar.bz2
$p3 = $p2->compress(Phar::NONE); // exception: /path/to/my.phar already
exists
?>
```

See Also

- Phar::compress()

PharData::compressFiles

PharData::compressFiles -- Compresses all files in the current tar/zip archive

Description

bool **PharData::compressFiles** (int `$compression`)

For tar-based archives, this method throws a BadMethodCallException, as compression of individual files within a tar archive is not supported by the file format. Use PharData::compress() to compress an entire tar-based archive.

For Zip-based archives, this method compresses all files in the archive using the specified compression. The zlib or bzip2 extensions must be enabled to take advantage of this feature. In addition, if any files are already compressed using bzip2/zlib compression, the respective extension must be enabled in order to decompress the files prior to re-compressing.

Parameters

`compression`
: Compression must be one of *Phar::GZ*, *Phar::BZ2* to add compression, or *Phar::NONE* to remove compression.

Errors/Exceptions

Throws BadMethodCallException if the phar.readonly INI variable is on, the zlib extension is not available, or if any files are compressed using bzip2 compression and the bzip2 extension is not enabled.

Examples

Example #62 - A PharData::compressFiles() example

```
<?php
$p = new Phar('/path/to/my.phar', 0, 'my.phar');
$p['myfile.txt'] = 'hi';
$p['myfile2.txt'] = 'hi';
foreach ($p as $file) {
   var_dump($file->getFileName());
   var_dump($file->isCompressed());
   var_dump($file->isCompressed(Phar::BZ2));
   var_dump($file->isCompressed(Phar::GZ));
}
$p->compressFiles(Phar::GZ);
foreach ($p as $file) {
```

```
    var_dump($file->getFileName());
    var_dump($file->isCompressed());
    var_dump($file->isCompressed(Phar::BZ2));
    var_dump($file->isCompressed(Phar::GZ));
}
?>
```

The above example will output:

```
string(10) "myfile.txt"
bool(false)
bool(false)
bool(false)
string(11) "myfile2.txt"
bool(false)
bool(false)
bool(false)
string(10) "myfile.txt"
int(4096)
bool(false)
bool(true)
string(11) "myfile2.txt"
int(4096)
bool(false)
bool(true)
```

See Also

- PharFileInfo::getCompressedSize()
- PharFileInfo::isCompressed()
- PharFileInfo::compress()
- PharFileInfo::decompress()
- Phar::canCompress()
- Phar::isCompressed()
- PharData::decompressFiles()
- Phar::getSupportedCompression()
- PharData::compress()
- PharData::decompress()

PharData::__construct

PharData::__construct -- Construct a non-executable tar or zip archive object

Description

void **PharData::__construct** (string `$fname` [, int `$flags`])

Parameters

`fname`
Path to an existing tar/zip archive or to-be-created archive

`flags`
flags to pass to Phar parent class RecursiveDirectoryIterator. See » SPL RecursiveDirectoryIterator docs

Errors/Exceptions

Throws BadMethodCallException if called twice, UnexpectedValueException if the phar archive can't be opened.

Examples

Example #63 - A PharData::__construct() example

```
<?php
try {
   $p = new PharData('/path/to/my.tar', CURRENT_AS_FILEINFO |
KEY_AS_FILENAME);
} catch (UnexpectedValueException $e) {
   die('Could not open my.tar');
} catch (BadMethodCallException $e) {
   echo 'technically, this cannot happen';
}
echo file_get_contents('phar:///path/to/my.tar/example.txt');
?>
```

PharData::convertToData

PharData::convertToData -- Convert a phar archive to a non-executable tar or zip file

Description

PharData **PharData::convertToData** ([int `$format` [, int `$compression` [, string `$extension`]]])

This method is used to convert a non-executable tar or zip archive to another non-executable format.

If no changes are specified, this method throws a BadMethodCallException. This method should be used to convert a tar archive to zip format or vice-versa. Although it is possible to simply change the compression of a tar archive using this method, it is better to use the PharData::compress() method for logical consistency.

If successful, the method creates a new archive on disk and returns a PharData object. The old archive is not removed from disk, and should be done manually after the process has finished.

Parameters

`format`
: This should be one of *Phar::TAR* or *Phar::ZIP*. If set to **NULL**, the existing file format will be preserved.

`compression`
: This should be one of *Phar::NONE* for no whole-archive compression, *Phar::GZ* for zlib-based compression, and *Phar::BZ2* for bzip-based compression.

`extension`
: This parameter is used to override the default file extension for a converted archive. Note that *.phar* cannot be used anywhere in the filename for a non-executable tar or zip archive. If converting to a tar-based phar archive, the default extensions are *.tar*, *.tar.gz*, and *.tar.bz2* depending on specified compression. For zip-based archives, the default extension is *.zip*.

Return Values

The method returns a PharData object on success and throws an exception on failure.

Errors/Exceptions

This method throws BadMethodCallException when unable to compress, an unknown compression method has been specified, the requested archive is buffering with **PharData::startBuffering()** and has not concluded with **PharData::stopBuffering()**, and

a PharException if any problems are encountered during the phar creation process.

Examples

Example #64 - A PharData::convertToData() example

Using PharData::convertToData():

```
<?php
try {
   $tarphar = new PharData('myphar.tar');
   // note that myphar.tar is *not* unlinked
   // convert it to the non-executable tar file format
   // creates myphar.zip
   $zip = $tarphar->convertToData(Phar::ZIP);
   // create myphar.tbz
   $tgz = $zip->convertToData(Phar::TAR, Phar::BZ2, '.tbz');
   // creates myphar.phar.tgz
   $phar = $tarphar->convertToData(Phar::PHAR); // throws exception
} catch (Exception $e) {
   // handle the error here
}
?>
```

See Also

- Phar::convertToExecutable()
- Phar::convertToData()
- PharData::convertToExecutable()

PharData::convertToExecutable

PharData::convertToExecutable -- Convert a non-executable tar/zip archive to an executable phar archive

Description

Phar **PharData::convertToExecutable** ([int `$format` [, int `$compression` [, string `$extension`]]])

Note
This method requires the *php.ini* setting *phar.readonly* to be set to *0* in order to work for Phar objects. Otherwise, a PharException will be thrown.

This method is used to convert a non-executable tar or zip archive to an executable phar archive. Any of the three executable file formats (phar, tar or zip) can be used, and whole-archive compression can also be performed.

If no changes are specified, this method throws a BadMethodCallException.

If successful, the method creates a new archive on disk and returns a Phar object. The old archive is not removed from disk, and should be done manually after the process has finished.

Parameters

`format`
: This should be one of *Phar::PHAR*, *Phar::TAR*, or *Phar::ZIP*. If set to **NULL**, the existing file format will be preserved.

`compression`
: This should be one of *Phar::NONE* for no whole-archive compression, *Phar::GZ* for zlib-based compression, and *Phar::BZ2* for bzip-based compression.

`extension`
: This parameter is used to override the default file extension for a converted archive. Note that all zip- and tar-based phar archives must contain*.phar* in their file extension in order to be processed as a phar archive. If converting to a phar-based archive, the default extensions are*.phar*,*.phar.gz*, or*.phar.bz2* depending on the specified compression. For tar-based phar archives, the default extensions are*.phar.tar*, *.phar.tar.gz*, and*.phar.tar.bz2*. For zip-based phar archives, the default extension is *.phar.zip*.

Return Values

The method returns a Phar object on success and throws an exception on failure.

Errors/Exceptions

This method throws BadMethodCallException when unable to compress, an unknown compression method has been specified, the requested archive is buffering with **PharData::startBuffering()** and has not concluded with **PharData::stopBuffering()**, an UnexpectedValueException if write support is disabled, and a PharException if any problems are encountered during the phar creation process.

Examples

Example #65 - A PharData::convertToExecutable() example

Using PharData::convertToExecutable():

```
<?php
try {
   $tarphar = new PharData('myphar.tar');
   // convert it to the phar file format
   // note that myphar.tar is *not* unlinked
   $phar = $tarphar->convertToExecutable(Phar::PHAR); // creates myphar.phar
   $phar->setStub($phar->createDefaultStub('cli.php', 'web/index.php'));
   // creates myphar.phar.tgz
   $compressed = $tarphar->convertToExecutable(Phar::TAR, Phar::GZ,
'.phar.tgz');
} catch (Exception $e) {
   // handle the error here
}
?>
```

See Also

- Phar::convertToExecutable()
- Phar::convertToData()
- PharData::convertToData()

PharData::copy

PharData::copy -- Copy a file internal to the phar archive to another new file within the phar

Description

bool **PharData::copy** (string `$oldfile`, string `$newfile`)

Copy a file internal to the tar/zip archive to another new file within the same archive. This is an object-oriented alternative to using copy() with the phar stream wrapper.

Parameters

`oldfile`

`newfile`

Return Values

returns **TRUE** on success, but it is safer to encase method call in a try/catch block and assume success if no exception is thrown.

Errors/Exceptions

throws UnexpectedValueException if the source file does not exist, the destination file already exists, write access is disabled, opening either file fails, reading the source file fails, or a PharException if writing the changes to the phar fails.

Examples

Example #66 - A PharData::copy() example

This example shows using PharData::copy() and the equivalent stream wrapper performance of the same thing. The primary difference between the two approaches is error handling. All PharData methods throw exceptions, whereas the stream wrapper uses trigger_error().

```
<?php
try {
   $phar = new PharData('myphar.tar');
   $phar['a'] = 'hi';
   $phar->copy('a', 'b');
```

```
   echo $phar['b']; // outputs "hi"
} catch (Exception $e) {
   // handle error
}

// the stream wrapper equivalent of the above code.
// E_WARNINGS are triggered on error rather than exceptions.
copy('phar://myphar.tar/a', 'phar//myphar.tar/c');
echo file_get_contents('phar://myphar.tar/c'); // outputs "hi"
?>
```

PharData::decompress

PharData::decompress -- Decompresses the entire Phar archive

Description

object **PharData::decompress** ([string `$extension`])

For tar-based archives, this method decompresses the entire archive.

For Zip-based archives, this method fails with an exception. The zlib extension must be enabled to decompress an archive compressed with with gzip compression, and the bzip2 extension must be enabled in order to decompress an archive compressed with bzip2 compression.

In addition, this method automatically renames the file extension of the archive,*.tar* by default. Alternatively, a file extension may be specified with the second parameter.

A PharData object is returned.

Parameters

`extension`
: For decompressing, the default file extension is*.phar.tar*. Use this parameter to specify another file extension. Be aware that no non-executable archives cannot contain*.phar* in their filename.

Errors/Exceptions

Throws BadMethodCallException if the zlib extension is not available, or the bzip2 extension is not enabled.

Examples

Example #67 - A PharData::decompress() example

```
<?php
$p = new PharData('/path/to/my.tar', 0, 'my.tar.gz');
$p['myfile.txt'] = 'hi';
$p['myfile2.txt'] = 'hi';
$p3 = $p2->decompress(); // creates /path/to/my.tar
?>
```

See Also

- PharFileInfo::getCompressedSize()
- PharFileInfo::isCompressed()
- PharFileInfo::compress()
- PharFileInfo::decompress()
- PharData::compress()
- Phar::canCompress()
- Phar::isCompressed()
- PharData::compress()
- Phar::getSupportedCompression()
- PharData::compressFiles()
- PharData::decompressFiles()

PharData::decompressFiles

PharData::decompressFiles -- Decompresses all files in the current zip archive

Description

bool **PharData::decompressFiles** (void)

Note
This method requires the *php.ini* setting *phar.readonly* to be set to *0* in order to work for Phar objects. Otherwise, a PharException will be thrown.

For tar-based archives, this method throws a BadMethodCallException, as compression of individual files within a tar archive is not supported by the file format. Use PharData::compress() to compress an entire tar-based archive.

For Zip-based archives, this method decompresses all files in the archive. The zlib or bzip2 extensions must be enabled to take advantage of this feature if any files are compressed using bzip2/zlib compression.

Errors/Exceptions

Throws BadMethodCallException if the zlib extension is not available, or if any files are compressed using bzip2 compression and the bzip2 extension is not enabled.

Examples

Example #68 - A PharData::decompressFiles() example

```
<?php
$p = new PharData('/path/to/my.zip');
$p['myfile.txt'] = 'hi';
$p['myfile2.txt'] = 'hi';
$p->compressFiles(Phar::GZ);
foreach ($p as $file) {
   var_dump($file->getFileName());
   var_dump($file->isCompressed());
   var_dump($file->isCompressed(Phar::BZ2));
   var_dump($file->isCompressed(Phar::GZ));
}
$p->decompressFiles();
foreach ($p as $file) {
   var_dump($file->getFileName());
   var_dump($file->isCompressed());
   var_dump($file->isCompressed(Phar::BZ2));
   var_dump($file->isCompressed(Phar::GZ));
```

```
}
?>
```

The above example will output:

```
string(10) "myfile.txt"
int(4096)
bool(false)
bool(true)
string(11) "myfile2.txt"
int(4096)
bool(false)
bool(true)
string(10) "myfile.txt"
bool(false)
bool(false)
bool(false)
string(11) "myfile2.txt"
bool(false)
bool(false)
bool(false)
```

See Also

- PharFileInfo::getCompressedSize()
- PharFileInfo::isCompressed()
- PharFileInfo::compress()
- PharFileInfo::decompress()
- Phar::canCompress()
- Phar::isCompressed()
- PharData::compressFiles()
- Phar::getSupportedCompression()
- PharData::compress()
- PharData::decompress()

PharData::delMetadata

PharData::delMetadata -- Deletes the global metadata of a zip archive

Description

int **PharData::delMetadata** (void)

Note
This method requires the *php.ini* setting *phar.readonly* to be set to *0* in order to work for Phar objects. Otherwise, a PharException will be thrown.

Deletes the global metadata of the zip archive

Parameters

Return Values

returns **TRUE** on success, but it is better to check for thrown exception, and assume success if none is thrown.

Errors/Exceptions

Throws PharException if errors occur while flushing changes to disk.

Examples

Example #69 - A PharData::delMetaData() example

```
<?php
try {
   $phar = new PharData('myphar.zip');
   var_dump($phar->getMetadata());
   $phar->setMetadata("hi there");
   var_dump($phar->getMetadata());
   $phar->delMetadata();
   var_dump($phar->getMetadata());
} catch (Exception $e) {
   // handle errors
}
?>
```

The above example will output:

```
NULL
string(8) "hi there"
NULL
```

See Also

- Phar::delMetadata()

PharData::delete

PharData::delete -- Delete a file within a tar/zip archive

Description

int **PharData::delete** (string `$entry`)

Delete a file within an archive. This is the functional equivalent of calling unlink() on the stream wrapper equivalent, as shown in the example below.

Parameters

`entry`
Path within an archive to the file to delete.

Return Values

returns **TRUE** on success, but it is better to check for thrown exception, and assume success if none is thrown.

Errors/Exceptions

Throws PharException if errors occur while flushing changes to disk.

Examples

Example #70 - A PharData::delete() example

```
<?php
try {
   $phar = new PharData('myphar.zip');
   $phar->delete('unlink/me.php');
   // this is equivalent to:
   unlink('phar://myphar.phar/unlink/me.php');
} catch (Exception $e) {
   // handle errors
}
?>
```

See Also

- Phar::delete()

PharData::extractTo

PharData::extractTo -- Extract the contents of a tar/zip archive to a directory

Description

int **PharData::extractTo** (string `$pathto` [, string|array `$files` [, bool `$overwrite`]])

Extract all files within a tar/zip archive to disk. Extracted files and directories preserve permissions as stored in the archive. The optional parameters allow optional control over which files are extracted, and whether existing files on disk can be overwritten. The second parameter *files* can be either the name of a file or directory to extract, or an array of names of files and directories to extract. By default, this method will not overwrite existing files, the third parameter can be set to true to enable overwriting of files. This method is similar to ZipArchive::extractTo().

Parameters

`pathto`
Path within an archive to the file to delete.

`files`
The name of a file or directory to extract, or an array of files/directories to extract

`overwrite`
FALSE by default, set to **TRUE** to enable overwriting existing files

Return Values

returns **TRUE** on success, but it is better to check for thrown exception, and assume success if none is thrown.

Errors/Exceptions

Throws PharException if errors occur while flushing changes to disk.

Examples

Example #71 - A PharData::extractTo() example

```
<?php
try {
   $phar = new PharData('myphar.tar');
   $phar->extractTo('/full/path'); // extract all files
   $phar->extractTo('/another/path', 'file.txt'); // extract only file.txt
```

```
   $phar->extractTo('/this/path',
       array('file1.txt', 'file2.txt')); // extract 2 files only
   $phar->extractTo('/third/path', null, true); // extract all files, and
overwrite
} catch (Exception $e) {
   // handle errors
}
?>
```

See Also

- Phar::extractTo()

PharData::isWritable

PharData::isWritable -- Returns true if the tar/zip archive can be modified

Description

bool **PharData::isWritable** (void)

This method returns **TRUE** if the tar/zip archive on disk is not read-only. Unlike Phar::isWritable(), data-only tar/zip archives can be modified even if *phar.readonly* is set to *1*.

Parameters

No parameters.

Return Values

Returns **TRUE** if the tar/zip archive can be modified

See Also

- Phar::canWrite()
- Phar::isWritable()

PharData::offsetSet

PharData::offsetSet -- set the contents of a file within the tar/zip to those of an external file or string

Description

void **PharData::offsetSet** (string `$offset`, string `$value`)

This is an implementation of the ArrayAccess interface allowing direct manipulation of the contents of a tar/zip archive using array access brackets. offsetSet is used for modifying an existing file, or adding a new file to a tar/zip archive.

Parameters

`offset`
: The filename (relative path) to modify in a tar or zip archive.

`value`
: Content of the file.

Return Values

No return values.

Errors/Exceptions

Throws PharException if there are any problems flushing changes made to the tar/zip archive to disk.

Examples

Example #72 - A PharData::offsetSet() example

offsetSet should not be accessed directly, but instead used via array access with the *[]* operator.

```
<?php
$p = new PharData('/path/to/my.tar');
try {
   // calls offsetSet
   $p['file.txt'] = 'Hi there';
} catch (Exception $e) {
   echo 'Could not modify file.txt:', $e;
}
```

```
?>
```

See Also

- Phar::offsetSet()

PharData::offsetUnset

PharData::offsetUnset -- remove a file from a tar/zip archive

Description

bool **PharData::offsetUnset** (string `$offset`)

This is an implementation of the ArrayAccess interface allowing direct manipulation of the contents of a tar/zip archive using array access brackets. offsetUnset is used for deleting an existing file, and is called by the unset() language construct.

Parameters

`offset`
The filename (relative path) to modify in the tar/zip archive.

Return Values

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

Errors/Exceptions

Throws PharException if there are any problems flushing changes made to the tar/zip archive to disk.

Examples

Example #73 - A PharData::offsetUnset() example

```
<?php
$p = new PharData('/path/to/my.zip');
try {
   // deletes file.txt from my.zip by calling offsetUnset
   unset($p['file.txt']);
} catch (Exception $e) {
   echo 'Could not delete file.txt: ', $e;
}
?>
```

See Also

- Phar::offsetUnset()

PharData::setAlias

PharData::setAlias -- dummy function (Phar::setAlias is not valid for PharData)

Description

bool **PharData::setAlias** (string `$alias`)

Non-executable tar/zip archives cannot have an alias, so this method simply throws an exception.

Parameters

`alias`
: A shorthand string that this archive can be referred to in *phar* stream wrapper access. This parameter is ignored.

Return Values

Errors/Exceptions

Throws PharException on all method calls

See Also

- Phar::setAlias()

PharData::setDefaultStub

PharData::setDefaultStub -- dummy function (Phar::setDefaultStub is not valid for PharData)

Description

void **PharData::setDefaultStub** ([string `$index` [, string `$webindex`]])

Non-executable tar/zip archives cannot have a stub, so this method simply throws an exception.

Parameters

`index`
: Relative path within the phar archive to run if accessed on the command-line

`webindex`
: Relative path within the phar archive to run if accessed through a web browser

Errors/Exceptions

Throws PharException on all method calls

See Also

- Phar::setDefaultStub()

Phar::setMetadata

Phar::setMetadata -- Sets phar archive meta-data

Description

void **Phar::setMetadata** (mixed `$metadata`)

Note
This method requires the *php.ini* setting *phar.readonly* to be set to *0* in order to work for Phar objects. Otherwise, a PharException will be thrown.

Phar::setMetadata() should be used to store customized data that describes something about the phar archive as a complete entity. PharFileInfo::setMetadata() should be used for file-specific meta-data. Meta-data can slow down the performance of loading a phar archive if the data is large.

Some possible uses for meta-data include specifying which file within the archive should be used to bootstrap the archive, or the location of a file manifest like » PEAR 's package.xml file. However, any useful data that describes the phar archive may be stored.

Parameters

`metadata`
Any PHP variable containing information to store that describes the phar archive

Examples

Example #74 - A Phar::setMetadata() example

```
<?php
// make sure it doesn't exist
@unlink('brandnewphar.phar');
try {
   $p = new Phar(dirname(__FILE__) . '/brandnewphar.phar', 0,
'brandnewphar.phar');
   $p['file.php'] = '<?php echo "hello"';
   $p->setMetadata(array('bootstrap' => 'file.php'));
   var_dump($p->getMetadata());
} catch (Exception $e) {
   echo 'Could not create and/or modify phar:', $e;
```

```
}
?>
```

The above example will output:

```
array(1) {
 ["bootstrap"]=>
 string(8) "file.php"
}
```

See Also

- Phar::getMetadata()
- Phar::delMetadata()
- Phar::hasMetadata()

Phar::setSignatureAlgorithm

Phar::setSignatureAlgorithm -- set the signature algorithm for a phar and apply it. The

Description

array **Phar::setSignatureAlgorithm** (int `$sigtype`)

Note
This method requires the *php.ini* setting *phar.readonly* to be set to *0* in order to work for Phar objects. Otherwise, a PharException will be thrown.

set the signature algorithm for a phar and apply it. The signature algorithm must be one of *Phar::MD5*, *Phar::SHA1*, *Phar::SHA256*, *Phar::SHA512*, or *Phar::PGP* (pgp not yet supported and falls back to SHA-1).

Parameters

`sigtype`
One of *Phar::MD5*, *Phar::SHA1*, *Phar::SHA256*, *Phar::SHA512*, or *Phar::PGP*

Return Values

Errors/Exceptions

Throws UnexpectedValueException for many errors, BadMethodCallException if called for a zip- or a tar-based phar archive, and a PharException if any problems occur flushing changes to disk.

See Also

- Phar::getSupportedSignatures()
- Phar::getSignature()

PharData::setStub

PharData::setStub -- dummy function (Phar::setStub is not valid for PharData)

Description

void **PharData::setStub** (string `$stub`)

Non-executable tar/zip archives cannot have a stub, so this method simply throws an exception.

Parameters

`stub`
: A string or an open stream handle to use as the executable stub for this phar archive. This parameter is ignored.

Errors/Exceptions

Throws PharException on all method calls

See Also

- Phar::setStub()

The PharFileInfo class

Introduction

The PharFileInfo class provides a high-level interface to the contents and attributes of a single file within a phar archive.

Class synopsis

PharFileInfo

PharFileInfo extends SplFileInfo {

/* Properties */

/* Methods */

void **PharFileInfo::chmod** (int `$permissions`)

bool **PharFileInfo::compress** (int `$compression`)

void **PharFileInfo::__construct** (string `$entry`)

bool **PharFileInfo::decompress** (void)

bool **PharFileInfo::delMetadata** (void)

int **PharFileInfo::getCRC32** (void)

int **PharFileInfo::getCompressedSize** (void)

int **PharFileInfo::getMetaData** (void)

int **PharFileInfo::getPharFlags** (void)

int **PharFileInfo::hasMetadata** (void)

bool **PharFileInfo::isCRCChecked** (void)

bool **PharFileInfo::isCompressed** (void)

bool **PharFileInfo::isCompressedBZIP2** (void)

bool **PharFileInfo::isCompressedGZ** (void)

bool **PharFileInfo::setCompressedBZIP2** (void)

bool **PharFileInfo::setCompressedGZ** (void)

void **PharFileInfo::setMetaData** (mixed `$metadata`)

bool **PharFileInfo::setUncompressed** (void)
}

PharFileInfo::chmod

PharFileInfo::chmod -- Sets file-specific permission bits

Description

void **PharFileInfo::chmod** (int `$permissions`)

PharFileInfo::chmod() allows setting of the executable file permissions bit, as well as read-only bits. Writeable bits are ignored, and set at runtime based on the phar.readonly INI variable. As with all functionality that modifies the contents of a phar, the phar.readonly INI variable must be off in order to succeed if the file is within a Phar archive. Files within PharData archives do not have this restriction.

Parameters

permissions
permissions (see chmod())

Examples

Example #75 - A PharFileInfo::chmod() example

```
<?php
// make sure it doesn't exist
@unlink('brandnewphar.phar');
try {
   $p = new Phar('brandnewphar.phar', 0, 'brandnewphar.phar');
   $p['file.sh'] = '#!/usr/local/lib/php
   <?php echo "hi"; ?>';
   // set executable bit
   $p['file.sh']->chmod(0555);
   var_dump($p['file.sh']->isExecutable());
} catch (Exception $e) {
   echo 'Could not create/modify phar: ', $e;
}
?>
```

The above example will output:

```
bool(true)
```

PharFileInfo::compress

PharFileInfo::compress -- Compresses the current Phar entry with either zlib or bzip2 compression

Description

bool **PharFileInfo::compress** (int `$compression`)

This method compresses the file inside the Phar archive using either bzip2 compression or zlib compression. The bzip2 or zlib extension must be enabled to take advantage of this feature. In addition, if the file is already compressed, the respective extension must be enabled in order to decompress the file. As with all functionality that modifies the contents of a phar, the phar.readonly INI variable must be off in order to succeed if the file is within a Phar archive. Files within PharData archives do not have this restriction.

Errors/Exceptions

Throws BadMethodCallException if the phar.readonly INI variable is on, or if the bzip2 / zlib extension is not available.

Examples

Example #76 - A PharFileInfo::compress() example

```
<?php
try {
   $p = new Phar('/path/to/my.phar', 0, 'my.phar');
   $p['myfile.txt'] = 'hi';
   $file = $p['myfile.txt'];
   var_dump($file->isCompressed(Phar::BZ2));
   $p['myfile.txt']->compress(Phar::BZ2);
   var_dump($file->isCompressed(Phar::BZ2));
} catch (Exception $e) {
   echo 'Create/modify operations on my.phar failed: ', $e;
}
?>
```

The above example will output:

```
bool(false)
bool(true)
```

See Also

- PharFileInfo::getCompressedSize()
- PharFileInfo::isCompressed()
- PharFileInfo::decompress()
- Phar::canCompress()
- Phar::isCompressed()
- Phar::compressFiles()
- Phar::decompressFiles()
- Phar::compress()
- Phar::decompress()
- Phar::getSupportedCompression()

PharFileInfo::__construct

PharFileInfo::__construct -- Construct a Phar entry object

Description

void **PharFileInfo::__construct** (string `$entry`)

This should not be called directly. Instead, a PharFileInfo object is initialized by calling Phar::offsetGet() through array access.

Parameters

`entry`
The full url to retrieve a file. If you wish to retrieve the information for the file *my/file.php* from the phar *boo.phar*, the entry should be *phar://boo.phar/my/file.php*.

Errors/Exceptions

Throws BadMethodCallException if **__construct()** is called twice. Throws UnexpectedValueException if the phar URL requested is malformed, the requested phar cannot be opened, or the file can't be found within the phar.

Examples

Example #77 - A PharFileInfo::__construct() example

```
<?php
try {
   $p = new Phar('/path/to/my.phar', 0, 'my.phar');
   $p['testfile.txt'] = "hi\nthere\ndude";
   $file = $p['testfile.txt'];
   foreach ($file as $line => $text) {
       echo "line number $line: $text";
   }
   // this also works
   $file = new PharFileInfo('phar:///path/to/my.phar/testfile.txt');
   foreach ($file as $line => $text) {
       echo "line number $line: $text";
   }
} catch (Exception $e) {
   echo 'Phar operations failed: ', $e;
}
?>
```

The above example will output:

```
line number 1: hi
line number 2: there
line number 3: dude
line number 1: hi
line number 2: there
line number 3: dude
```

PharFileInfo::decompress

PharFileInfo::decompress -- Decompresses the current Phar entry within the phar

Description

bool **PharFileInfo::decompress** (void)

This method decompresses the file inside the Phar archive. Depending on how the file is compressed, the bzip2 or zlib extensions must be enabled to take advantage of this feature. As with all functionality that modifies the contents of a phar, the phar.readonly INI variable must be off in order to succeed if the file is within a Phar archive. Files within PharData archives do not have this restriction.

Errors/Exceptions

Throws BadMethodCallException if the phar.readonly INI variable is on, or if the bzip2 / zlib extension is not available.

Examples

Example #78 - A PharFileInfo::decompress() example

```
<?php
try {
   $p = new Phar('/path/to/my.phar', 0, 'my.phar');
   $p['myfile.txt'] = 'hi';
   $file = $p['myfile.txt'];
   $file->compress(Phar::GZ);
   var_dump($file->isCompressed());
   $p['myfile.txt']->decompress();
   var_dump($file->isCompressed());
} catch (Exception $e) {
   echo 'Create/modify failed for my.phar: ', $e;
}
?>
```

The above example will output:

```
int(4096)
bool(false)
```

See Also

- PharFileInfo::getCompressedSize()

- PharFileInfo::isCompressed()
- PharFileInfo::compress()
- Phar::canCompress()
- Phar::isCompressed()
- Phar::compressFiles()
- Phar::decompressFiles()
- Phar::compress()
- Phar::decompress()
- Phar::getSupportedCompression()

PharFileInfo::delMetadata

PharFileInfo::delMetadata -- Deletes the metadata of the entry

Description

bool **PharFileInfo::delMetadata** (void)

Deletes the metadata of the entry, if any.

Parameters

No parameters.

Return Values

Returns **TRUE** if successful, **FALSE** if the entry had no metadata. As with all functionality that modifies the contents of a phar, the phar.readonly INI variable must be off in order to succeed if the file is within a Phar archive. Files within PharData archives do not have this restriction.

Errors/Exceptions

Throws PharException if errors occurred while flushing changes to disk, and BadMethodCallException if write access is disabled.

Examples

Example #79 - A PharFileInfo::delMetaData() example

```
<?php
try {
   $a = new Phar('myphar.phar');
   $a['hi'] = 'hi';
   var_dump($a['hi']->delMetadata());
   $a['hi']->setMetadata('there');
   var_dump($a['hi']->delMetadata());
   var_dump($a['hi']->delMetadata());
} catch (Exception $e) {
   // handle errors
}
?>
```

The above example will output:

```
bool(false)
bool(true)
bool(false)
```

See Also

- PharFileInfo::setMetadata()
- PharFileInfo::hasMetadata()
- PharFileInfo::getMetadata()
- Phar::setMetadata()
- Phar::hasMetadata()
- Phar::getMetadata()

PharFileInfo::getCRC32

PharFileInfo::getCRC32 -- Returns CRC32 code or throws an exception if CRC has not been verified

Description

int **PharFileInfo::getCRC32** (void)

This returns the crc32() checksum of the file within the Phar archive.

Return Values

The crc32() checksum of the file within the Phar archive.

Errors/Exceptions

Throws BadMethodCallException if the file has not yet had its CRC32 verified. This should be impossible with normal use, as the CRC is verified upon opening the file for reading or writing.

Examples

Example #80 - A PharFileInfo::getCRC32() example

```
<?php
try {
   $p = new Phar('/path/to/my.phar', 0, 'my.phar');
   $p['myfile.txt'] = 'hi';
   $file = $p['myfile.txt'];
   echo $file->getCRC32();
} catch (Exception $e) {
   echo 'Write operations on my.phar.phar failed: ', $e;
}
?>
```

The above example will output:

```
3633523372
```

PharFileInfo::getCompressedSize

PharFileInfo::getCompressedSize -- Returns the actual size of the file (with compression) inside the Phar archive

Description

int **PharFileInfo::getCompressedSize** (void)

This returns the size of the file within the Phar archive. Uncompressed files will return the same value for getCompressedSize as they will with filesize()

Return Values

The size in bytes of the file within the Phar archive on disk.

Examples

Example #81 - A PharFileInfo::getCompressedSize() example

```
<?php
try {
   $p = new Phar('/path/to/my.phar', 0, 'my.phar');
   $p['myfile.txt'] = 'hi';
   $file = $p['myfile.txt'];
   echo $file->getCompressedSize();
} catch (Exception $e) {
   echo 'Write operations failed on my.phar: ', $e;
}
?>
```

The above example will output:

```
2
```

See Also

- PharFileInfo::isCompressed()
- PharFileInfo::decompress()
- PharFileInfo::compress()
- Phar::canCompress()
- Phar::isCompressed()
- Phar::compress()
- Phar::decompress()

- Phar::getSupportedCompression()
- Phar::decompressFiles()
- Phar::compressFiles()

PharFileInfo::getMetaData

PharFileInfo::getMetaData -- Returns file-specific meta-data saved with a file

Description

int **PharFileInfo::getMetaData** (void)

Return meta-data that was saved in the Phar archive's manifest for this file.

Parameters

Return Values

any PHP variable that can be serialized and is stored as meta-data for the file, or **NULL** if no meta-data is stored.

Examples

Example #82 - A PharFileInfo::getMetaData() example

```
<?php
// make sure it doesn't exist
@unlink('brandnewphar.phar');
try {
   $p = new Phar(dirname(__FILE__) . '/brandnewphar.phar', 0,
'brandnewphar.phar');
   $p['file.txt'] = 'hello';
   $p['file.txt']->setMetaData(array('user' => 'bill', 'mime-type' =>
'text/plain'));
   var_dump($p['file.txt']->getMetaData());
} catch (Exception $e) {
   echo 'Could not create/modify brandnewphar.phar: ', $e;
}
?>
```

The above example will output:

```
array(2) {
 ["user"]=>
 string(4) "bill"
 ["mime-type"]=>
 string(10) "text/plain"
}
```

See Also

- PharFileInfo::setMetadata()
- PharFileInfo::hasMetadata()
- PharFileInfo::delMetadata()
- Phar::setMetadata()
- Phar::hasMetadata()
- Phar::getMetadata()

PharFileInfo::getPharFlags

PharFileInfo::getPharFlags -- Returns the Phar file entry flags

Description

int **PharFileInfo::getPharFlags** (void)

This returns the flags set in the manifest for a Phar. This will always return *0* in the current implementation.

Return Values

The Phar flags (always *0* in the current implementation)

Examples

Example #83 - A PharFileInfo::getPharFlags() example

```
<?php
try {
   $p = new Phar('/path/to/my.phar', 0, 'my.phar');
   $p['myfile.txt'] = 'hi';
   $file = $p['myfile.txt'];
   var_dump($file->getPharFlags());
} catch (Exception $e) {
   echo 'Could not create/modify my.phar: ', $e;
}
?>
```

The above example will output:

```
int(0)
```

PharFileInfo::hasMetadata

PharFileInfo::hasMetadata -- Returns the metadata of the entry

Description

int **PharFileInfo::hasMetadata** (void)

Returns the metadata of a file within a phar archive.

Parameters

No parameters.

Return Values

Returns **FALSE** if no metadata is set or is **NULL**, **TRUE** if metadata is not **NULL**

See Also

- PharFileInfo::setMetadata()
- PharFileInfo::getMetadata()
- PharFileInfo::delMetadata()
- Phar::setMetadata()
- Phar::hasMetadata()
- Phar::getMetadata()

PharFileInfo::isCRCChecked

PharFileInfo::isCRCChecked -- Returns whether file entry has had its CRC verified

Description

bool **PharFileInfo::isCRCChecked** (void)

This returns whether a file within a Phar archive has had its CRC verified.

Return Values

TRUE if the file has had its CRC verified, **FALSE** if not.

Examples

Example #84 - A PharFileInfo::isCRCChecked() example

```
<?php
try {
   $p = new Phar('/path/to/my.phar', 0, 'my.phar');
   $p['myfile.txt'] = 'hi';
   $file = $p['myfile.txt'];
   var_dump($file->isCRCChecked());
} catch (Exception $e) {
   echo 'Create/modify operations failed on my.phar: ', $e;
}
?>
```

The above example will output:

```
bool(true)
```

PharFileInfo::isCompressed

PharFileInfo::isCompressed -- Returns whether the entry is compressed

Description

bool **PharFileInfo::isCompressed** (void)

This returns whether a file is compressed within a Phar archive with either Gzip or Bzip2 compression.

Return Values

TRUE if the file is compressed within the Phar archive, **FALSE** if not.

Examples

Example #85 - A PharFileInfo::isCompressed() example

```
<?php
try {
   $p = new Phar('/path/to/my.phar', 0, 'my.phar');
   $p['myfile.txt'] = 'hi';
   $p['myfile2.txt'] = 'hi';
   $p['myfile2.txt']->setCompressedGZ();
   $file = $p['myfile.txt'];
   $file2 = $p['myfile2.txt'];
   var_dump($file->isCompressed());
   var_dump($file2->isCompressed());
} catch (Exception $e) {
   echo 'Create/modify on phar my.phar failed: ', $e;
}
?>
```

The above example will output:

```
bool(false)
bool(true)
```

See Also

- PharFileInfo::getCompressedSize()
- PharFileInfo::decompress()
- PharFileInfo::compress()
- Phar::decompress()

- Phar::compress()
- Phar::canCompress()
- Phar::isCompressed()
- Phar::getSupportedCompression()
- Phar::decompressFiles()
- Phar::compressFiles()

PharFileInfo::isCompressedBZIP2

PharFileInfo::isCompressedBZIP2 -- Returns whether the entry is compressed using bzip2

Description

bool **PharFileInfo::isCompressedBZIP2** (void)

Note
This method has been removed from the phar extension as of version 2.0.0. Alternative implementations are available using PharFileInfo::isCompressed(), PharFileInfo::decompress(), and PharFileInfo::compress().

This returns whether a file is compressed within a Phar archive with Bzip2 compression.

Return Values

TRUE if the file is compressed within the Phar archive using Bzip2, **FALSE** if not.

Examples

Example #86 - A PharFileInfo::isCompressedBZIP2() example

```
<?php
try {
   $p = new Phar('/path/to/my.phar', 0, 'my.phar');
   $p['myfile.txt'] = 'hi';
   $p['myfile2.txt'] = 'hi';
   $p['myfile3.txt'] = 'hi';
   $p['myfile2.txt']->setCompressedGZ();
   $p['myfile3.txt']->setCompressedBZIP2();
   $file = $p['myfile.txt'];
   $file2 = $p['myfile2.txt'];
   $file3 = $p['myfile3.txt'];
   var_dump($file->isCompressedBZIP2());
   var_dump($file2->isCompressedBZIP2());
   var_dump($file3->isCompressedBZIP2());
} catch (Exception $e) {
   echo 'Create/modify on phar my.phar failed: ', $e;
}
?>
```

The above example will output:

```
bool(false)
bool(false)
bool(true)
```

See Also

- PharFileInfo::getCompressedSize()
- PharFileInfo::isCompressed()
- PharFileInfo::isCompressedGZ()
- PharFileInfo::setCompressedBZIP2()
- PharFileInfo::setUncompressed()
- PharFileInfo::setCompressedGZ()
- Phar::canCompress()
- Phar::isCompressed()
- Phar::compressAllFilesBZIP2()
- Phar::compressAllFilesGZ()
- Phar::getSupportedCompression()
- Phar::uncompressAllFiles()

PharFileInfo::isCompressedGZ

PharFileInfo::isCompressedGZ -- Returns whether the entry is compressed using gz

Description

bool **PharFileInfo::isCompressedGZ** (void)

Note
This method has been removed from the phar extension as of version 2.0.0. Alternative implementations are available using PharFileInfo::isCompressed(), PharFileInfo::decompress(), and PharFileInfo::compress().

This returns whether a file is compressed within a Phar archive with Gzip compression.

Return Values

TRUE if the file is compressed within the Phar archive using Gzip, **FALSE** if not.

Examples

Example #87 - A PharFileInfo::isCompressedGZ() example

```
<?php
try {
   $p = new Phar('/path/to/my.phar', 0, 'my.phar');
   $p['myfile.txt'] = 'hi';
   $p['myfile2.txt'] = 'hi';
   $p['myfile3.txt'] = 'hi';
   $p['myfile2.txt']->setCompressedGZ();
   $p['myfile3.txt']->setCompressedBZIP2();
   $file = $p['myfile.txt'];
   $file2 = $p['myfile2.txt'];
   $file3 = $p['myfile3.txt'];
   var_dump($file->isCompressedGZ());
   var_dump($file2->isCompressedGZ());
   var_dump($file3->isCompressedGZ());
} catch (Exception $e) {
   echo 'Create/modify on phar my.phar failed: ', $e;
}
?>
```

The above example will output:

```
bool(false)
bool(true)
bool(false)
```

See Also

- PharFileInfo::getCompressedSize()
- PharFileInfo::isCompressedBZIP2()
- PharFileInfo::isCompressed()
- PharFileInfo::setCompressedBZIP2()
- PharFileInfo::setUncompressed()
- PharFileInfo::setCompressedGZ()
- Phar::canCompress()
- Phar::isCompressed()
- Phar::compressAllFilesBZIP2()
- Phar::compressAllFilesGZ()
- Phar::getSupportedCompression()
- Phar::uncompressAllFiles()

PharFileInfo::setCompressedBZIP2

PharFileInfo::setCompressedBZIP2 -- Compresses the current Phar entry within the phar using Bzip2 compression

Description

bool **PharFileInfo::setCompressedBZIP2** (void)

Note
This method has been removed from the phar extension as of version 2.0.0. Alternative implementations are available using PharFileInfo::isCompressed(), PharFileInfo::decompress(), and PharFileInfo::compress().

This method compresses the file inside the Phar archive using bzip2 compression. The bzip2 extension must be enabled to take advantage of this feature. In addition, if the file is already compressed using gzip compression, the zlib extension must be enabled in order to decompress the file. As with all functionality that modifies the contents of a phar, the phar.readonly INI variable must be off in order to succeed.

Errors/Exceptions

Throws BadMethodCallException if the phar.readonly INI variable is on, or if the bzip2 extension is not available.

Examples

Example #88 - A PharFileInfo::setCompressedBZIP2() example

```
<?php
try {
   $p = new Phar('/path/to/my.phar', 0, 'my.phar');
   $p['myfile.txt'] = 'hi';
   $file = $p['myfile.txt'];
   var_dump($file->isCompressedBZIP2());
   $p['myfile.txt']->setCompressedBZIP2();
   var_dump($file->isCompressedBZIP2());
} catch (Exception $e) {
   echo 'Create/modify operations on my.phar failed: ', $e;
}
?>
```

The above example will output:

```
bool(false)
bool(true)
```

See Also

- PharFileInfo::getCompressedSize()
- PharFileInfo::isCompressedBZIP2()
- PharFileInfo::isCompressed()
- PharFileInfo::isCompressedGZ()
- PharFileInfo::setUncompressed()
- PharFileInfo::setCompressedGZ()
- Phar::canCompress()
- Phar::isCompressed()
- Phar::compressAllFilesBZIP2()
- Phar::compressAllFilesGZ()
- Phar::getSupportedCompression()
- Phar::uncompressAllFiles()

PharFileInfo::setCompressedGZ

PharFileInfo::setCompressedGZ -- Compresses the current Phar entry within the phar using gz compression

Description

bool **PharFileInfo::setCompressedGZ** (void)

Note
This method has been removed from the phar extension as of version 2.0.0. Alternative implementations are available using PharFileInfo::isCompressed(), PharFileInfo::decompress(), and PharFileInfo::compress().

This method compresses the file inside the Phar archive using gzip compression. The zlib extension must be enabled to take advantage of this feature. In addition, if the file is already compressed using bzip2 compression, the bzip2 extension must be enabled in order to decompress the file. As with all functionality that modifies the contents of a phar, the phar.readonly INI variable must be off in order to succeed.

Errors/Exceptions

Throws BadMethodCallException if the phar.readonly INI variable is on, or if the zlib extension is not available.

Examples

Example #89 - A PharFileInfo::setCompressedGZ() example

```
<?php
try {
   $p = new Phar('/path/to/my.phar', 0, 'my.phar');
   $p['myfile.txt'] = 'hi';
   $file = $p['myfile.txt'];
   var_dump($file->isCompressedGZ());
   $p['myfile.txt']->setCompressedGZ();
   var_dump($file->isCompressedGZ());
} catch (Exception $e) {
   echo 'Create/modify operations on my.phar failed: ', $e;
}
?>
```

The above example will output:

```
bool(false)
bool(true)
```

See Also

- PharFileInfo::getCompressedSize()
- PharFileInfo::isCompressedBZIP2()
- PharFileInfo::isCompressed()
- PharFileInfo::isCompressedGZ()
- PharFileInfo::setCompressedBZIP2()
- PharFileInfo::setUncompressed()
- Phar::canCompress()
- Phar::isCompressed()
- Phar::compressAllFilesBZIP2()
- Phar::compressAllFilesGZ()
- Phar::getSupportedCompression()
- Phar::uncompressAllFiles()

PharFileInfo::setMetaData

PharFileInfo::setMetaData -- Sets file-specific meta-data saved with a file

Description

void **PharFileInfo::setMetaData** (mixed `$metadata`)

setMetaData() should only be used to store customized data in a file that cannot be represented with existing information stored with a file. Meta-data can significantly slow down the performance of loading a phar archive if the data is large, or if there are many files containing meta-data. It is important to note that file permissions are natively supported inside a phar; it is possible to set them with the PharFileInfo::chmod() method. As with all functionality that modifies the contents of a phar, the phar.readonly INI variable must be off in order to succeed if the file is within a Phar archive. Files within PharData archives do not have this restriction.

Some possible uses for meta-data include passing a user/group that should be set when a file is extracted from the phar to disk. Other uses could include explicitly specifying a MIME type to return. However, any useful data that describes a file, but should not be contained inside of it may be stored.

Parameters

`metadata`
Any PHP variable containing information to store alongside a file

Examples

Example #90 - A PharFileInfo::setMetaData() example

```
<?php
// make sure it doesn't exist
@unlink('brandnewphar.phar');
try {
   $p = new Phar(dirname(__FILE__) . '/brandnewphar.phar', 0,
'brandnewphar.phar');
   $p['file.txt'] = 'hello';
   $p['file.txt']->setMetaData(array('user' => 'bill', 'mime-type' =>
'text/plain'));
   var_dump($p['file.txt']->getMetaData());
} catch (Exception $e) {
   echo 'Could not create/modify phar: ', $e;
}
?>
```

The above example will output:

```
array(2) {
 ["user"]=>
 string(4) "bill"
 ["mime-type"]=>
 string(10) "text/plain"
}
```

See Also

- PharFileInfo::hasMetadata()
- PharFileInfo::getMetadata()
- PharFileInfo::delMetadata()
- Phar::setMetadata()
- Phar::hasMetadata()
- Phar::getMetadata()

PharFileInfo::setUncompressed

PharFileInfo::setUncompressed -- Uncompresses the current Phar entry within the phar, if it is compressed

Description

bool **PharFileInfo::setUncompressed** (void)

Note
This method has been removed from the phar extension as of version 2.0.0. Alternative implementations are available using PharFileInfo::isCompressed(), PharFileInfo::decompress(), and PharFileInfo::compress().

This method decompresses the file inside the Phar archive. Depending on how the file is compressed, the bzip2 or zlib extensions must be enabled to take advantage of this feature. As with all functionality that modifies the contents of a phar, the phar.readonly INI variable must be off in order to succeed.

Errors/Exceptions

Throws BadMethodCallException if the phar.readonly INI variable is on, or if the bzip2 / zlib extension is not available.

Examples

Example #91 - A PharFileInfo::setUncompressed() example

```
<?php
try {
   $p = new Phar('/path/to/my.phar', 0, 'my.phar');
   $p['myfile.txt'] = 'hi';
   $file = $p['myfile.txt'];
   $file->setCompressedGZ();
   var_dump($file->isCompressed());
   $p['myfile.txt']->setUncompressed();
   var_dump($file->isCompressed());
} catch (Exception $e) {
   echo 'Create/modify failed for my.phar: ', $e;
}
?>
```

The above example will output:

```
bool(true)
bool(false)
```

See Also

- PharFileInfo::getCompressedSize()
- PharFileInfo::isCompressedBZIP2()
- PharFileInfo::isCompressed()
- PharFileInfo::isCompressedGZ()
- PharFileInfo::setCompressedBZIP2()
- PharFileInfo::setCompressedGZ()
- Phar::canCompress()
- Phar::isCompressed()
- Phar::compressAllFilesBZIP2()
- Phar::compressAllFilesGZ()
- Phar::getSupportedCompression()
- Phar::uncompressAllFiles()

The PharException class

Introduction

The PharException class provides a phar-specific exception class for try/catch blocks.

Class synopsis

PharException

```
PharException extends Exception {

    /*  Properties */
}
```

PharException

PharException -- The PharException class provides a phar-specific exception class for try/catch blocks.

Description