StreamManager

Release v0.1.1

Oct 08, 2017

Contents

A manager for PHP streams, using callback functions.

- Source repository
- Continuous Integration
- API documentation

Contents:

CHAPTER 1

Introduction

This projects originates from the following facts:

- PHP streams form a powerful toolbox, with oft-underrated features (custom streams, custom filters, etc.)
- The PHP approach to streams interactions closely follows that of C, to the point that the function names are often the same (`fread()`, `fwrite()`, etc.)

  Developpers who are not accustomed to C may find it difficult to work with PHP streams (eg. dealing with partial reads/writes).
- Though powerful, the API can be quite difficult to master for beginners or even seasoned developers. The implementation also comes with several annoying limitations,

  For example, the `stream_select()` function is very useful when working with non-blocking streams, until you start using stream filters and realize that it does not work for filtered strems.

  This can lead to confusion or even frustration on the part of PHP developers.
- Last but not least, combining multiple projects each using streams can prove to be a real challenge (eg. each project expects to be able to run an endless `stream_select()` loop, leaving no way for other libraries to do their share of the job)

The PHP stream manager was created as an attempt to get rid of some of these hindrances by providing:

- A common framework for libraries to build upon, so that multiple projects can be used together without fear of conflicts related to stream management.
- A simple approach to stream management, using callback functions to handle various events (incoming data, disconnections, ...)
- An abstraction layer that adds support for filtered streams in `stream_select()`, while still retaining the original PHP stream API (eg. so that things like `stream_filter_append()` still work the same way)

CHAPTER 2

Installation

The PHP Stream Manager relies on Composer for its installation.

Use the following command to install the manager:

```
$ php composer.php require fpoirotte/stream-manager
```

CHAPTER 3

Usage

The manager's event loop

To use the manager, you first need to create an instance for it:

```
use fpoirotte\StreamManager;

$manager = new StreamManager();
```

You can then create a new PHP stream like you would normally, and pass it to the manager, with a label for later reference:

```
// Create a new stream and store it in the manager
// under the label "rot13".
$rawStream  = fopen('php://temp/', 'w+');
$manager['rot13']   = $rawStream;
```

When passed to the manager like this, the stream will actually be wrapped in a new special stream. This new stream is what is actually stored in the manager using the given label.

Any operation after that point should be done on the wrapper rather than on the original (raw) stream. The wrapper can be used to add PHP stream filters, write/read data, and so on.

```
// Add a filter so that any data written to the stream
// gets automatically scrambled using the ROT13 encoding scheme.
stream_filter_append($manager['rot13'], 'string.rot13', STREAM_FILTER_WRITE);

// Write some data to the stream (wrapper).
fwrite($manager['rot13'], "Hello world!");
```

Once the streams have all been registered, we tell the manager to do its job until there are no longer any stream to manage:

```
$manager->loop();
```

Note: Alternatively, it is possible to control exactly how many times the manager will run its event loop, by giving it a maximum iteration count:

```
$manager->loop(10);
```

An iteration occurs after an event is received or an error occurs. The manager will run in a loop until the given number of iterations is reached or there are no more streams left to manage. By default, the number of expected iterations is 0, which causes the manager to loop endlessly.

Events and callback functions

The manager relies on callback functions to process certains events. It also defines default callback functions to handle those events.

The following callback functions are used:

- `readCallback`: this function is called when the stream has incoming data pending a read.

  There is no default implementation for this callback. Depending on the type of stream used, some data may be dropped when PHP's internal buffer is filled, or the stream may block until some of the data has been read.

- `closeCallback`: this function is called when the stream is closed by another party (such as a network socket being disconnected by its peer).

  The default implementation simply closes the stream and removes it from list of streams currently handled by the stream manager.

To change the callback associated with an event, use the following code snippet.

Note: To have any effect, these options must be set **before** the raw stream is registered with the manager.

Also, some versions of HHVM do not support the use of array options with `stream_context_set_option()`. Thus, it is advised that you set each option separately using the long key-value call form.

```
// Call "onDataReceived" when new data is received.
stream_context_set_option($rawStream, StreamManager::WRAPPER_NAME, 'readCallback',
↪'onDataReceived');

// Call "onStreamClosed" when the stream gets closed by its peer.
stream_context_set_option($rawStream, StreamManager::WRAPPER_NAME, 'closeCallback',
↪'onStreamClosed');
```

Each callback will be called with 3 arguments:

- The instance of the manager responsible for triggering the event (ie. an instance of `fpoirotte\\StreamManager`)
- The instance of the stream wrapper associated with the event, which can be used with stream-related PHP functions (`fread()`, `fwrite()`, `fclose()`, and so on)
- The label attached to the stream wrapper during its registration with the stream manager. This label can be used, for example, to remove the stream from the list of streams currently handled by the stream manager.

The following code snippet is an example of what a `closeCallback` might look like:

```php
function onStreamClosed($manager, $stream, $name)
{
    // Close the stream on our side too.
    @fclose($stream);

    // Remove the stream from the list of streams handled by the manager.
    unset($manager[$name]);
}
```

Badges: