Library Documentation

---

1. Introduction

1.1 Overview

PHP CSV Utilities is a fully object-oriented PHP5 library for working with CSV files. Although PHP has native support for dealing with CSV files, they can be a bit painful to work with. The goal of this library is to ease that pain as much as possible.

Because of the relatively large name of this library, I'll use its full name, *PHP CSV Utilities*, as well as it's abbreviation, *PCU*, interchangeably throughout this documentation and beyond.

1.2 Inspiration

PCU was inspired by python's native csv module and borrows heavily from it in both interface as well as philosophy. Although I stayed true to its interface where I could, there are some areas that just didn't

translate well to PHP and were given a different interface in this library. There are also components in this library that are unique and you will not find their counterpart in python's csv module. Basically I used python's csv module as a starting point.

1.3 Philosophy

The philosophy of this library is modularity. Every class in this library is not only capable of being extended, but encouraged. While designing PHP CSV Utilities, I have followed industry-standard OOP design patterns and techniques to create a library that is easy to extend and customize to fit your needs.

1.2 Installation

1.2.1 Downloading & extracting

You may download the library at the PHP CSV Utilities Google Code page. Here you will see (on the right) a "featured downloads" section which will contain the newest release. If you need an older release, click on the "downloads" tab above.

The library comes in two archive formats: a tarball (.tar.gz) or a zip file (.zip). Which format you choose is up to you (if you don't know which you need, use the zip file).

To extract the archive(s) on Windows:

To extract the tarball, you'll need to get an archiving tool capable of gzip compression. Winzip or WinRar are both capable of this.

To extract the zip file, double click the file and then click "extract all files". Then follow the instructions on the extractin wizard.

To extract the archive(s) on Linux (replace <filename> with the path to the archive):

To extract the tarball:

```
tar xvzf <filename>.tar.gz
```

To extract the zip:

```
tar xvzf <filename>.zip
```

1.2.2 Uploading to your server

After extracting the library files, you should now have a folder called "pcu". Inside you will see the following:

```
/pcu
    /Csv
    /docs
    /tests
```

You only need to upload the “Csv” directory to your web server. It is recommended that you upload them into a directory that is in your php include_path.

1.2.3 Running the unit tests

To run the unit tests, upload both “test” and “Csv” directories. Go to “tests/index.php” in your web browser, or if you have command-line access to your server, open it in your cli. Either method will result in a report telling you (hopefully) that the tests all pass.

---

2. Topics

2.1 Getting Started

By default, both Csv_Reader and Csv_Writer will assume a format similar to what is produced by Excel when exporting to a csv file. If your files are in a different format, skip ahead to section 2.4 “Using dialects” to learn to configure your readers and writers properly. If you are just learning how to use the library and need some sample data to work with, you can find some sample files in the “tests/data” folder.

For the topics below, consider this table when “products.csv” is referenced:

id	name	price	description	taxable
1	Widget	10.99	A wonderful wittle widget.	1
2	Whatsamahoozit	1.99	The best Whatsamahoozit this side of Wyoming.	0
3	Dandy Doodad	19.99	This is one dandy doodad.	1
4	Thingamajigger	100	Thingamajiggers are the best product known to man.	1
5	Jolly Junk	.99	This is just some junk.	1
6	Something	40.49	I like this. It is something. It isn’t taxable.	0

2.2 Reading a csv file

For this, you’ll need to instantiate a new Csv_Reader object. Pass the path to your CSV file as the first argument to the object.

```
$reader = new Csv_Reader('/path/to/data/products.csv');
```

Now, because Csv_Reader implements the Iterator SPL Interface, you can loop through it with a foreach loop, just like an array. The following will print a list of product names:

```
$reader = new Csv_Reader('/path/to/data/products.csv');
foreach ($reader as $row) {
    print $row[1] . "<br>";
}
```

Other methods of looping through a file

The following will print the first 10 rows in the CSV file:

```
$reader = new Csv_Reader('/path/to/data/products.csv');
$i = 0;
while ($row = $reader->getRow() && $i < 10) {
    print $row[1] . "<br>";
    $i++;
}
```

Like I said before, Csv_Reader implements the SPL Iterable interface. This means the standard iterable methods are available as well:

```
$reader = new Csv_Reader('/path/to/data/products.csv');
while ($row = $reader->current()) {
    print $row[1] . "<br>";
    $reader->next(); // advances the internal pointer
}
```

Convert a CSV file to an array

Although this whole object-oriented interface stuff is cool, sometimes you just need to work with an array.

```
$rows = $reader->toArray();
```

Determine how many rows are in a CSV file

Both of the following methods will give you the total rows in the file:

```
echo count($reader);
echo $reader->count();
```

2.3 Writing / appending a csv file

Instantiate a new Csv_Writer object and pass it the path to your CSV file.

```
$writer = new Csv_Writer('/path/to/data/products.csv');
```

From here there are several methods of writing rows to the CSV file. If you need to write to the file line by line, you can use the writeRow() method.

```
$writer = new Csv_Writer('/path/to/data/products.csv');
$writer->writeRow(array(10, 'Chicken Gizzards', 9.99, 'Some prime chicken gizzards',
1));
```

It is just as simple to write multiple rows to the file with writeRows().

```
$writer = new Csv_Writer('/path/to/data/products.csv');
$rows = array(
    array(10, 'And another thing', 90, 'This is just another thing', 0),
```

```
    array(11, 'Bacon', 8.99, 'Delicious', 1),
    array(12, 'Cheap Bacon', .99, 'Some cheap bacon', 1),
);
$writer->writeRows($rows);
```

Appending a CSV file

Csv_Writer also accepts a file handle in its constructor which means you can open a file in append mode, pass it to Csv_Writer and it will append rather than overwrite the file.

```
$products_file = fopen('/path/to/data/products.csv', 'a');
$writer = new Csv_Writer($products_file);
$writer->writeRow(array(10, 'Chicken Gizzards', 9.99, 'Some prime chicken gizzards',
1));
```

2.4 Using dialects

To provide a common interface for specifying CSV file format to your readers and writers, PCU comes with a class called Csv_Dialect. This class makes it very simple to specify the format you'd like to read or write. The library comes with several dialects you can use right out of the box. Because there really is no definitive CSV format, the default dialect (used when the user doesn't specify one) is what I consider to be the most common CSV format.

Reformat a CSV file

Just to show you how cool dialects are, and how easy they make things, Let's assume that products.csv is comma-delimited and all columns are quoted. I'm going to reformat the file to use tabs and quote only if there are special characters in the column.

```
$reader = new Csv_Reader('products.csv', new Csv_Dialect_Excel);
$dialect = new Csv_Dialect_Excel();
$dialect->quoting = Csv_Dialect::QUOTE_MINIMAL;
$dialect->delimiter = "\t";
$writer = new Csv_Writer('new-products.csv', $dialect);
$writer->writeRows($reader);
```

2.5 Auto-detecting csv format

2.6 Mapping

2.7 Error Handling

---

3. Components

3.1 Csv_Reader

This component provides a very simple interface for reading csv files of just about any format. The constructor takes either a file name or a file resource as its first argument. Once instantiated, there are several methods for reading the file. You can use a while loop, a for loop, even a foreach loop. Or, if you prefer, you can convert the whole file to a php array.

Csv_Reader is completely configurable. As its second parameter, it accepts what is called a dialect (a Csv_Dialect object) that defines its format.

3.2 Csv_Writer

This component provides a very simple interface for writing csv files in just about any format. You may write row by row, or you can pass it a two-dimensional array and it will write the whole thing to a csv file.

Csv_Writer is completely configurable. As its second parameter, it accepts what is called a dialect (a Csv_Dialect object) that defines the format in which the file is to be written.

3.4 Csv_Dialect

One of the most complicated issues with CSV is that there really is no standard format for it. Although CSV stands for "comma-separated values", there is no guarantee a csv file will actually be separated by commas. CSV files vary wildly from file to file, application to application. Some use commas, some use tabs, some use different line endings, or different quoting styles, the list goes on. PCU solves this problem by allowing the user to provide a Csv_Dialect class to many of its components which tells them which format the CSV file is or should be. If a dialect is not provided, a reasonable default will be used.

3.4 Csv_Sniffer

This component inspects a sample of CSV data and attempts to deduce its format. It returns a Csv_Dialect object which you can then pass to your readers and writers. It can also attempt to detect if there is a header row.

This class will probably be renamed in the near future. I am considering all of these class names at the moment: Csv_Detect, Csv_AutoDetect Csv_Inspect, Csv_Probe, Csv_Reflect.

3.5 Csv_Mapper

Generally, when reading a CSV file with Csv_Reader, you will get a numerically indexed array for each row you read. Many CSV files have what is called a header row that gives a name to each column in the file. Csv_Mapper maps this header row (or a user-provided array) to its columns so you that the reader provides you with an associative array. This way you can refer to each column by its name.

3.6 Csv_Exception

PCU will throw exceptions any time there is an exceptional situation. You should learn to work with exceptions if you haven’t done so before.