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

Before you will be able to run the unit tests, you'll need to download the SimpleTest PHP unit testing library and place it somewhere on your include path (the easiest way would be to place the "simpletest" directory inside of the library's "tests" directory). After this, make sure the tests/tmp directory is writable.

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 attempt to deduce the format of the CSV file automatically. If you'd like to explicitly specify the 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 (including the header row):

```
$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();
```

Specifying whether the file has a header row

By default, Csv_Reader will not attempt to detect whether or not your CSV file has a header row. It will simply assume it doesn't and return numerically indexed rows as shown above. If you know your file has a header row, you may specify this and Csv_Reader's methods will, from that point on, return associative arrays, indexed by the first row:

```
$reader = new Csv_Reader('/path/to/products.csv');
$reader->useFirstRowAsHeader();
$row = $reader->getRow();
echo $row['name']; // return arrays will now use header as keys
```

Note: when using this method, Csv_Reader will essentially ignore your header row from the time you call useFirstRowAsHeader(). If you need to retrieve this row at any point after calling that method, use getHeader().

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

```
$reader->useFirstRowAsHeader();
$header = $reader->getHeader();
```

If you aren't sure whether or not your file has a header row, simply call Csv_Reader::detectHasHeader() to determine this.

```
$reader = new Csv_Reader('/path/to/products.csv');
if ($reader->detectHasHeader()) {
    $reader->useFirstRowAsHeader();
}
```

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));
```

Explicitly specifying a header row

If you'd like to specify the row that should be used as the header row in the CSV file, simply pass an array of column names to setHeader() and this will guarantee that the first row in the file will be your header row.

$writer->setHeader(array('id','name','price','description','taxable'));

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.

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

Auto-detecting the format of a CSV file in PCU is incredibly easy. Csv_Reader attempts to detect the format of a csv file by default.

```
$reader = new Csv_Reader('products.csv');
$row1 = $reader->getRow(); // will most likely already know the format
```

If you do not want Csv_Reader to detect the file's format, pass a Csv_Dialect object as the second parameter to its constructor.

```
$dialect = new Csv_Reader('/path/to/products.csv');
$dialect->delimiter = “\t”; // use a tab as the delimiter
$dialect->lineterminator = “\n”; // use a standard newline as the line terminator
$dialect->quoting = Csv_Dialect::QUOTE_ALL; // quote all columns
$dialect->escapechar = “\””; // escape quotes with a quote character
$dialect->quotechar = “\””; // use double quotes to quote
```

Alternately, you may pass an array with all of these values to Csv_Dialect's constructor instead of setting the properties manually:

```
$dialect = new Csv_Dialect(array(
    'delimiter' => “\t”,
    'lineterminator' => “\n”,
    'quoting' => Csv_Dialect::QUOTE_ALL,
));
```

2.6 Error Handling

PCU throws exceptions when it encounters any type of problem. Such problems include files not be readable or writable, or perhaps not existing as well as the inability to determine a file's format. To see where exceptions are thrown in PCU, see the component documentation below.

For example, when attempting to read a CSV file, you can wrap your code in a try/catch block:

```
try {
    $reader = new Csv_Reader('/path/to/products.csv');
} catch (Csv_Exception_FileNotFound $e) {
    echo "Sorry, the file you requested could not be found.";
}
```

To forgo the use of exceptions, use the following:

```
try {
    // put CSV code in here and exceptions will essentially be ignored
} catch (Csv_Exception $e) {
}
```

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 the format in which the file is to be written. If a Csv_Dialect object is not passed to the Csv_Reader object, it will attempt to deduce the format of the file automatically.

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.

3.4 Csv_ AutoDetect

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.

3.5 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.