

## PHPExcel Developer Documentation

**Author:** Maarten Balliauw  
**Version:** 1.7.9  
**Date:** 02 June 2013

# 1. Contents

PHPExcel Developer Documentation .....	1
1. Contents .....	2
2. Prerequisites .....	4
2.1. Software requirements .....	4
2.2. Installation instructions .....	4
2.3. Getting started .....	4
2.4. Useful links and tools .....	4
2.4.1. OpenXML / SpreadsheetML .....	4
2.4.2. Frequently asked questions .....	5
2.4.3. Tutorials .....	6
3. Architecture .....	7
3.1. Schematical .....	7
3.2. Lazy Loader .....	7
3.3. Spreadsheet in memory .....	7
3.4. Readers and writers .....	7
3.5. Fluent interfaces .....	8
4. Creating a spreadsheet .....	10
4.1. The PHPEXcel class .....	10
4.1.1. Loading a Workbook from a file .....	10
4.1.2. Creating a new workbook .....	10
4.2. Configuration Settings .....	10
4.2.1. Cell Caching .....	10
4.2.2. Language/Locale .....	12
4.3. Clearing a Workbook from memory .....	13
4.4. Worksheets .....	13
4.4.1. Adding a new Worksheet .....	14
4.4.2. Copying Worksheets .....	14
4.4.3. Removing a Worksheet .....	14
4.5. Accessing cells .....	14
4.5.1. Setting a cell value by coordinate .....	14
4.5.2. Retrieving a cell by coordinate .....	15
4.5.3. Setting a cell value by column and row .....	15
4.5.4. Retrieving a cell by column and row .....	15
4.5.5. Looping cells .....	15
4.5.6. Using value binders to facilitate data entry .....	16
4.6. PHPEXcel recipes .....	17
4.6.1. Setting a spreadsheet's metadata .....	17
4.6.2. Setting a spreadsheet's active sheet .....	17
4.6.3. Write a date or time into a cell .....	17
4.6.4. Write a formula into a cell .....	18
4.6.5. Locale Settings for Formulae .....	19
4.6.6. Write a newline character "\n" in a cell (ALT+"Enter") .....	20
4.6.7. Explicitly set a cell's datatype .....	20
4.6.8. Change a cell into a clickable URL .....	20
4.6.9. Setting a worksheet's page orientation and size .....	20
4.6.10. Page Setup: Scaling options .....	21
4.6.11. Page margins .....	22
4.6.12. Center a page horizontally/vertically .....	22
4.6.13. Setting the print header and footer of a worksheet .....	22
4.6.14. Setting printing breaks on a row or column .....	24
4.6.15. Show/hide gridlines when printing .....	24
4.6.16. Setting rows/columns to repeat at top/left .....	24
4.6.17. Specify printing area .....	24
4.6.18. Formatting cells .....	24
4.6.19. Number formats .....	26
4.6.20. Alignment and wrap text .....	26
4.6.21. Setting the default style of a workbook .....	27

4.6.22.	Styling cell borders.....	27
4.6.23.	Conditional formatting a cell.....	28
4.6.24.	Add a comment to a cell .....	28
4.6.25.	Apply autofilter to a range of cells .....	29
4.6.26.	Setting security on a spreadsheet .....	29
4.6.27.	Setting data validation on a cell.....	30
4.6.28.	Setting a column's width.....	30
4.6.29.	Show/hide a column .....	31
4.6.30.	Group/outline a column.....	31
4.6.31.	Setting a row's height .....	31
4.6.32.	Show/hide a row.....	31
4.6.33.	Group/outline a row .....	32
4.6.34.	Merge/unmerge cells .....	32
4.6.35.	Inserting rows/columns.....	32
4.6.36.	Add a drawing to a worksheet .....	32
4.6.37.	Reading Images from a worksheet.....	33
4.6.38.	Add rich text to a cell .....	34
4.6.39.	Define a named range .....	34
4.6.40.	Redirect output to a client's web browser .....	34
4.6.41.	Setting the default column width .....	35
4.6.42.	Setting the default row height .....	35
4.6.43.	Add a GD drawing to a worksheet .....	35
4.6.44.	Setting worksheet zoom level .....	36
4.6.45.	Sheet tab color.....	36
4.6.46.	Creating worksheets in a workbook .....	36
4.6.47.	Hidden worksheets (Sheet states) .....	36
4.6.48.	Right-to-left worksheet .....	36
5.	Performing formula calculations .....	37
5.1.	Using the PHPExcel calculation engine .....	37
5.2.	Known limitations .....	38
5.2.1.	Operator precedence .....	38
5.2.2.	Formulas involving numbers and text.....	38
6.	Reading and writing to file .....	39
6.1.	PHPExcel_IOFactory.....	39
6.1.1.	Creating PHPExcel_Reader_IReader using PHPExcel_IOFactory.....	39
6.1.2.	Creating PHPExcel_Writer_IWriter using PHPExcel_IOFactory .....	39
6.2.	Excel 2007 (SpreadsheetML) file format.....	39
6.2.1.	PHPExcel_Reader_Excel2007 .....	40
6.2.2.	PHPExcel_Writer_Excel2007.....	40
6.3.	Excel 5 (BIFF) file format .....	41
6.3.1.	PHPExcel_Reader_Excel5 .....	41
6.3.2.	PHPExcel_Writer_Excel5 .....	42
6.4.	Excel 2003 XML file format .....	42
6.4.1.	PHPExcel_Reader_Excel2003XML .....	42
6.5.	Symbolic Link (SYLK).....	43
6.5.1.	PHPExcel_Reader_SYLK .....	43
6.6.	Open/Libre Office (.ods).....	43
6.6.1.	PHPExcel_Reader_OOCalc .....	43
6.7.	CSV (Comma Separated Values).....	44
6.7.1.	PHPExcel_Reader_CSV .....	44
6.7.2.	PHPExcel_Writer_CSV.....	45
6.8.	HTML.....	46
6.8.1.	PHPExcel_Reader_HTML.....	46
6.8.2.	PHPExcel_Writer_HTML .....	46
6.9.	PDF .....	47
6.9.1.	PHPExcel_Writer_PDF .....	47
6.10.	Generating Excel files from templates (read, modify, write) .....	49
7.	Credits .....	50
Appendix A:	Valid array keys for style applyFromArray() .....	51

## 2. Prerequisites

### 2.1. Software requirements

The following software is required to develop using PHPEXcel:

- » PHP version 5.2.0 or newer
- » PHP extension `php_zip` enabled \*)
- » PHP extension `php_xml` enabled
- » PHP extension `php_gd2` enabled (if not compiled in)

\*) `php_zip` is only needed by `PHPExcel_Reader_Excel2007`, `PHPExcel_Writer_Excel2007` and `PHPExcel_Reader_OOCalc`. In other words, if you need PHPEXcel to handle `.xlsx` or `.ods` files you will need the zip extension, but otherwise not.  
You can remove this dependency for writing Excel2007 files (though not yet for reading) by using the PCLZip library that is bundled with PHPEXcel. See the FAQ section of this document (2.4.2) for details about this. PCLZip does have a dependency on PHP's `zlib` extension being enabled.

### 2.2. Installation instructions

Installation is quite easy: copy the contents of the Classes folder to any location within your application source directories.

*Example:*

If your web root folder is `/var/www/` you may want to create a subfolder called `/var/www/Classes/` and copy the files into that folder so you end up with files:

```
/var/www/Classes/PHPExcel.php  
/var/www/Classes/PHPExcel/Calculation.php  
/var/www/Classes/PHPExcel/Cell.php  
...
```

### 2.3. Getting started

A good way to get started is to run some of the tests included in the download.

Copy the "Examples" folder next to your "Classes" folder from above so you end up with:

```
/var/www/ Examples/01simple.php  
/var/www/ Examples/02types.php  
...
```

Start running the tests by pointing your browser to the test scripts:

```
http://example.com/ Examples/01simple.php  
http://example.com/ Examples/02types.php  
...
```

Note: It may be necessary to modify the `include/require` statements at the beginning of each of the test scripts if your "Classes" folder from above is named differently.

### 2.4. Useful links and tools

There are some links and tools which are very useful when developing using PHPEXcel. Please refer to the [PHPExcel CodePlex pages](#) for an update version of the list below.

#### 2.4.1. OpenXML / SpreadsheetML

- » File format documentation  
[http://www.ecma-international.org/news/TC45\\_current\\_work/TC45\\_available\\_docs.htm](http://www.ecma-international.org/news/TC45_current_work/TC45_available_docs.htm)
- » OpenXML Explained e-book  
<http://openxmldeveloper.org/articles/1970.aspx>
- » Microsoft Office Compatibility Pack for Word, Excel, and PowerPoint 2007 File Formats  
<http://www.microsoft.com/downloads/details.aspx?familyid=941b3470-3ae9-4aee-8f43->

[c6bb74cd1466&displaylang=en](http://www.codeplex.com/PHPExcel/PackageExplorer/)

- » **OpenXML Package Explorer**  
<http://www.codeplex.com/PHPExcel/PackageExplorer/>

## 2.4.2. Frequently asked questions

The up-to-date F.A.Q. page for PHPEXcel can be found on <http://www.codeplex.com/PHPExcel/Wiki/View.aspx?title=FAQ&referringTitle=Requirements>.

### There seems to be a problem with character encoding...

It is necessary to use UTF-8 encoding for all texts in PHPEXcel. If the script uses different encoding then you can convert those texts with PHP's `iconv()` or `mb_convert_encoding()` functions.

### PHP complains about ZipArchive not being found

Make sure you meet all requirements, especially `php_zip` extension should be enabled.

The `ZipArchive` class is only required when reading or writing formats that use Zip compression (Excel2007 and OOCalc). Since version 1.7.6 the PCLZip library has been bundled with PHPEXcel as an alternative to the `ZipArchive` class.

This can be enabled by calling:

```
PHPExcel_Settings::setZipClass(PHPEXcel_Settings::PCLZIP);
```

*before* calling the save method of the Excel2007 Writer.

You can revert to using `ZipArchive` by calling:

```
PHPExcel_Settings::setZipClass(PHPEXcel_Settings::ZIPARCHIVE);
```

At present, this only allows you to write Excel2007 files without the need for `ZipArchive` (not to read Excel2007 or OOCalc)

### Excel 2007 cannot open the file generated by PHPEXcel\_Writer\_2007 on Windows

*"Excel found unreadable content in '\*.xlsx'. Do you want to recover the contents of this workbook? If you trust the source of this workbook, click Yes."*

Some older versions of the 5.2.x `php_zip` extension on Windows contain an error when creating ZIP files. The version that can be found on <http://snaps.php.net/win32/php5.2-win32-latest.zip> should work at all times.

Alternatively, upgrading to at least PHP 5.2.9 should solve the problem.

If you can't locate a clean copy of `ZipArchive`, then you can use the PCLZip library as an alternative when writing Excel2007 files, as described above.

### Fatal error: Allowed memory size of xxx bytes exhausted (tried to allocate yyy bytes) in zzz on line aaa

PHPExcel holds an "in memory" representation of a spreadsheet, so it is restricted by PHP's memory limitations. The memory made available to PHP can be increased by editing the value of the `memory_limit` directive in your `php.ini` file, or by using `ini_set('memory_limit', '128M')` within your code (ISP permitting).

Some Readers and Writers are faster than others, and they also use differing amounts of memory. You can find some indication of the relative performance and memory usage for the different Readers and Writers, over the different versions of PHPEXcel, on the [discussion board](#).

If you've already increased memory to a maximum, or can't change your memory limit, then [this discussion](#) on the board describes some of the methods that can be applied to reduce the memory usage of your scripts using PHPEXcel.

### Protection on my worksheet is not working?

When you make use of any of the worksheet protection features (e.g. cell range protection, prohibiting deleting rows, ...), make sure you enable worksheet security. This can for example be done like this:

```
$objPHPExcel->getActiveSheet()->getProtection()->setSheet(true);
```

### Feature X is not working with PHPEXcel\_Reader\_Y / PHPEXcel\_Writer\_Z

Not all features of PHPEXcel are implemented in all of the Reader / Writer classes. This is mostly due to underlying libraries not supporting a specific feature or not having implemented a specific feature.

For example autofilter is not implemented in PEAR Spreadsheet\_Excel\_writer, which is the base of our Excel5 writer.

We are slowly building up a list of features, together with the different readers and writers that support them, in the "Functionality Cross-Reference.xls" file in the /Documentation folder.

### Formulas don't seem to be calculated in Excel2003 using compatibility pack?

This is normal behaviour of the compatibility pack, Excel2007 displays this correctly. Use PHPEXcel\_Writer\_Excel5 if you really need calculated values, or force recalculation in Excel2003.

### Setting column width is not 100% accurate

Trying to set column width, I experience one problem. When I open the file in Excel, the actual width is 0.71 less than it should be.

The short answer is that PHPEXcel uses a measure where padding is included. See section: "Setting a column's width" for more details.

### How do I use PHPEXcel with my framework

- » There are some instructions for using PHPEXcel with Joomla on the [Joomla message board](#)
- » A page of advice on using [PHPEXcel in the Yii framework](#)
- » [The Bakery](#) has some helper classes for reading and writing with PHPEXcel within CakePHP
- » Integrating [PHPEXcel into Kohana](#) <http://www.flynsarmy.com/2010/07/phpexcel-module-for-kohana-3/> and [Интеграция PHPEXcel и Kohana Framework](#)
- » Using [PHPEXcel with TYPO3](#)

### Joomla Autoloader interferes with PHPEXcel Autoloader

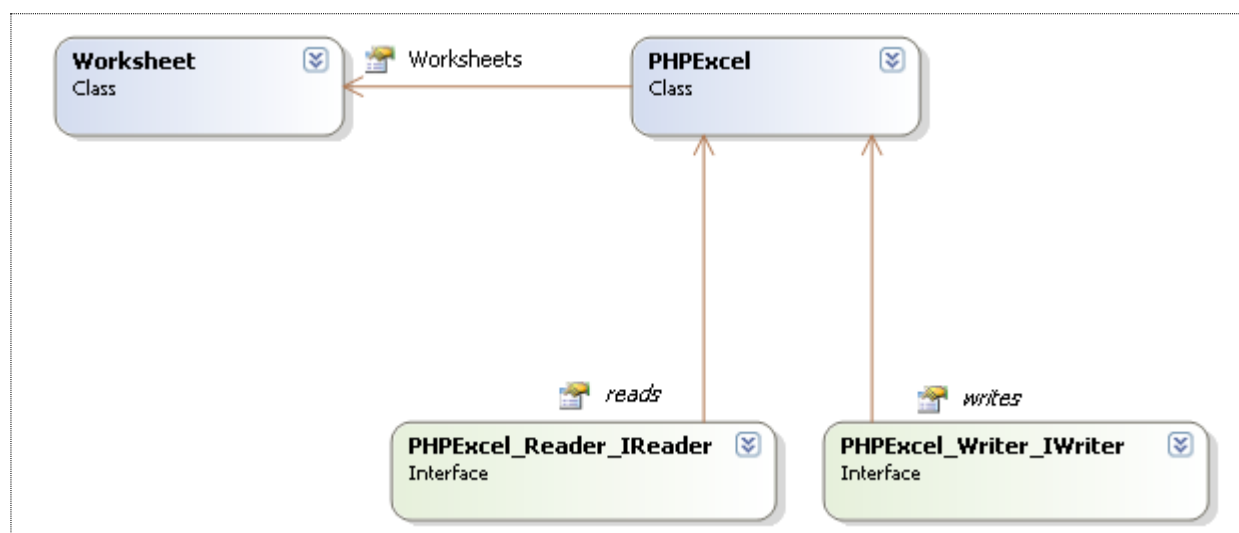
Thanks to peterrlynch for the following advice on resolving issues between the [PHPEXcel autoloader](#) and [Joomla Autoloader](#)

## 2.4.3. Tutorials

- » English PHPEXcel tutorial  
<http://openxmldeveloper.org>
- » French PHPEXcel tutorial  
<http://g-ernaelsten.developpez.com/tutoriels/excel2007/>
- » Russian PHPEXcel Blog Postings  
<http://www.web-junior.net/sozdanie-excel-fajjlov-s-pomoshhyu-phpexcel/>
- » A Japanese-language introduction to PHPEXcel  
<http://journal.mycom.co.jp/articles/2009/03/06/phpexcel/index.html>

## 3. Architecture

### 3.1. Schematical



### 3.2. Lazy Loader

PHPExcel implements an autoloader or “lazy loader”, which means that it is not necessary to include every file within PHPEXcel. It is only necessary to include the initial PHPEXcel class file, then the autoloader will include other class files as and when required, so only those files that are actually required by your script will be loaded into PHP memory. The main benefit of this is that it reduces the memory footprint of PHPEXcel itself, so that it uses less PHP memory.

If your own scripts already define an autoload function, then this may be overwritten by the PHPEXcel autoload function. For example, if you have:

```
function __autoload($class) {
    ...
}
```

Do this instead:

```
function myAutoload($class) {
    ...
}
spl_autoload_register('myAutoload');
```

Your autoloader will then co-exist with the autoloader of PHPEXcel.

### 3.3. Spreadsheet in memory

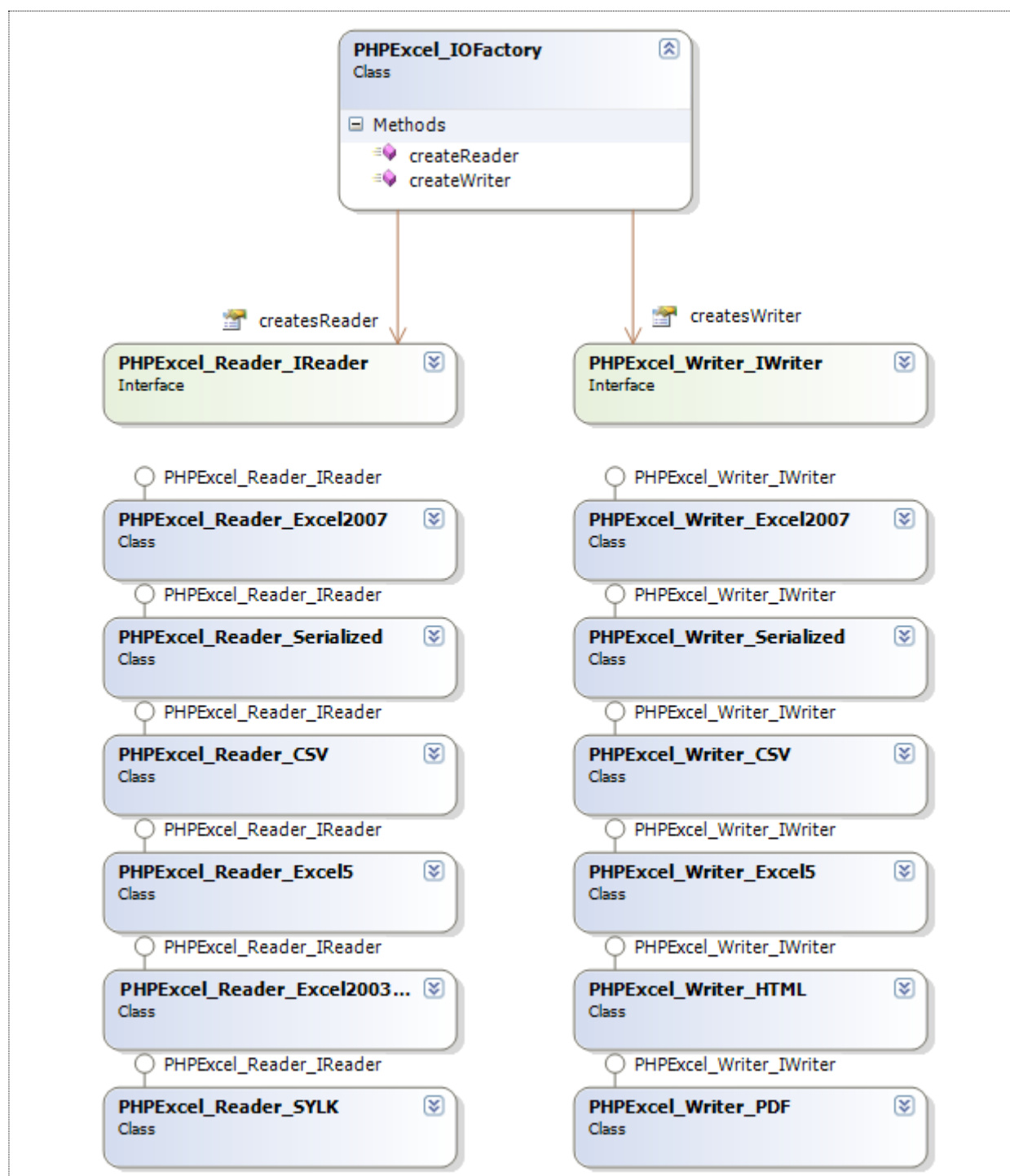
PHPExcel’s architecture is built in a way that it can serve as an in-memory spreadsheet. This means that, if one would want to create a web based view of a spreadsheet which communicates with PHPEXcel’s object model, he would only have to write the front-end code.

Just like desktop spreadsheet software, PHPEXcel represents a spreadsheet containing one or more worksheets, which contain cells with data, formulas, images, ...

### 3.4. Readers and writers

On its own, PHPEXcel does not provide the functionality to read from or write to a persisted spreadsheet (on disk or in a database). To provide that functionality, readers and writers can be used.

By default, the PHPEXcel package provides some readers and writers, including one for the Open XML spreadsheet format (a.k.a. Excel 2007 file format). You are not limited to the default readers and writers, as you are free to implement the PHPEXcel\_Reader\_IReader and PHPEXcel\_Writer\_IWriter interface in a custom class.



### 3.5. *Fluent interfaces*

PHPExcel supports fluent interfaces in most locations. This means that you can easily “chain” calls to specific methods without requiring a new PHP statement. For example, take the following code:

```
$objPHPExcel->getProperties()->setCreator("Maarten Balliauw");
$objPHPExcel->getProperties()->setLastModifiedBy("Maarten Balliauw");
```

```
$objPHPExcel->getProperties()->setTitle("Office 2007 XLSX Test Document");  
$objPHPExcel->getProperties()->setSubject("Office 2007 XLSX Test Document");  
$objPHPExcel->getProperties()->setDescription("Test document for Office 2007 XLSX,  
generated using PHP classes.");  
$objPHPExcel->getProperties()->setKeywords("office 2007 openxml php");  
$objPHPExcel->getProperties()->setCategory("Test result file");
```

This can be rewritten as:

```
$objPHPExcel->getProperties()  
    ->setCreator("Maarten Balliauw")  
    ->setLastModifiedBy("Maarten Balliauw")  
    ->setTitle("Office 2007 XLSX Test Document")  
    ->setSubject("Office 2007 XLSX Test Document")  
    ->setDescription("Test document for Office 2007 XLSX, generated using  
PHP classes.")  
    ->setKeywords("office 2007 openxml php")  
    ->setCategory("Test result file");
```

**❗ Using fluent interfaces is not required**

Fluent interfaces have been implemented to provide a convenient programming API. Use of them is not required, but can make your code easier to read and maintain. It can also improve performance, as you are reducing the overall number of calls to PHPEXcel methods.

## 4. Creating a spreadsheet

### 4.1. The PHPEXcel class

The PHPEXcel class is the core of PHPEXcel. It contains references to the contained worksheets, document security settings and document meta data.

To simplify the PHPEXcel concept: the PHPEXcel class represents your workbook. Typically, you will create a workbook in one of two ways, either by loading it from a spreadsheet file, or creating it manually. A third option, though less commonly used, is cloning an existing workbook that has been created using one of the previous two methods.

#### 4.1.1. Loading a Workbook from a file

Details of the different spreadsheet formats supported, and the options available to read them into a PHPEXcel object are described fully in the “PHPExcel User Documentation - Reading Spreadsheet Files” document.

```
$inputFileName = './sampleData/example1.xls';

/** Load $inputFileName to a PHPEXcel Object */
$objPHPExcel = PHPEXcel_IOFactory::load($inputFileName);
```

#### 4.1.2. Creating a new workbook

If you want to create a new workbook, rather than load one from file, then you simply need to instantiate it as a new PHPEXcel object.

```
/** Create a new PHPEXcel Object */
$objPHPExcel = new PHPEXcel();
```

A new workbook will always be created with a single worksheet.

## 4.2. Configuration Settings

Once you have included the PHPEXcel files in your script, but before instantiating a PHPEXcel object or loading a workbook file, there are a number of configuration options that can be set which will affect the subsequent behaviour of the script.

### 4.2.1. Cell Caching

PHPExcel uses an average of about 1k/cell in your worksheets, so large workbooks can quickly use up available memory. Cell caching provides a mechanism that allows PHPEXcel to maintain the cell objects in a smaller size of memory, on disk, or in APC, memcache or Wincache, rather than in PHP memory. This allows you to reduce the memory usage for large workbooks, although at a cost of speed to access cell data.

By default, PHPEXcel still holds all cell objects in memory, but you can specify alternatives. To enable cell caching, you must call the PHPEXcel\_Settings::setCacheStorageMethod() method, passing in the caching method that you wish to use.

```
$cacheMethod = PHPEXcel_CachedObjectStorageFactory::cache_in_memory;
PHPExcel_Settings::setCacheStorageMethod($cacheMethod);
```

setCacheStorageMethod() will return a boolean true on success, false on failure (for example if trying to cache to APC when APC is not enabled).

A separate cache is maintained for each individual worksheet, and is automatically created when the worksheet is instantiated based on the caching method and settings that you have configured. You cannot change the configuration settings once you have started to read a workbook, or have created your first worksheet.

Currently, the following caching methods are available.

<code>PHPExcel_CachedObjectStorageFactory::cache_in_memory;</code>
The default. If you don't initialise any caching method, then this is the method that PHPEXcel will use. Cell objects are maintained in PHP memory as at present.
<code>PHPExcel_CachedObjectStorageFactory::cache_in_memory_serialized;</code>
Using this caching method, cells are held in PHP memory as an array of serialized objects, which reduces the memory footprint with minimal performance overhead.
<code>PHPExcel_CachedObjectStorageFactory::cache_in_memory_gzip;</code>
Like <code>cache_in_memory_serialized</code> , this method holds cells in PHP memory as an array of serialized objects, but gzipped to reduce the memory usage still further, although access to read or write a cell is slightly slower.
<code>PHPExcel_CachedObjectStorageFactory::cache_igbinary;</code>
Uses PHP's igbinary extension (if it's available) to serialize cell objects in memory. This is normally faster and uses less memory than standard PHP serialization, but isn't available in most hosting environments.
<code>PHPExcel_CachedObjectStorageFactory::cache_to_discISAM;</code>
When using <code>cache_to_discISAM</code> all cells are held in a temporary disk file, with only an index to their location in that file maintained in PHP memory. This is slower than any of the <code>cache_in_memory</code> methods, but significantly reduces the memory footprint. By default, PHPEXcel will use PHP's temp directory for the cache file, but you can specify a different directory when initialising <code>cache_to_discISAM</code> . <div> <pre>\$cacheMethod = PHPEXcel_CachedObjectStorageFactory:: cache_to_discISAM; \$cacheSettings = array( 'dir' =&gt; '/usr/local/tmp' ); PHPExcel_Settings::setCacheStorageMethod(\$cacheMethod, \$cacheSettings);</pre> </div>
The temporary disk file is automatically deleted when your script terminates.
<code>PHPExcel_CachedObjectStorageFactory::cache_to_phpTemp;</code>
Like <code>cache_to_discISAM</code> , when using <code>cache_to_phpTemp</code> all cells are held in the <code>php://temp</code> I/O stream, with only an index to their location maintained in PHP memory. In PHP, the <code>php://memory</code> wrapper stores data in the memory: <code>php://temp</code> behaves similarly, but uses a temporary file for storing the data when a certain memory limit is reached. The default is 1 MB, but you can change this when initialising <code>cache_to_phpTemp</code> . <div> <pre>\$cacheMethod = PHPEXcel_CachedObjectStorageFactory:: cache_to_phpTemp; \$cacheSettings = array( 'memoryCacheSize' =&gt; '8MB' ); PHPExcel_Settings::setCacheStorageMethod(\$cacheMethod, \$cacheSettings);</pre> </div>
The <code>php://temp</code> file is automatically deleted when your script terminates.
<code>PHPExcel_CachedObjectStorageFactory::cache_to_apc;</code>
When using <code>cache_to_apc</code> , cell objects are maintained in APC <sup>1</sup> with only an index maintained in PHP memory to identify that the cell exists. By default, an APC cache timeout of 600 seconds is used, which should be enough for most applications: although it is possible to change this when initialising <code>cache_to_APC</code> . <div> <pre>\$cacheMethod = PHPEXcel_CachedObjectStorageFactory::cache to APC;</pre> </div>

<sup>1</sup> You must have APC enabled for PHP to use this option.

```
$cacheSettings = array( 'cacheTime'      => 600
                       );
PHPExcel_Settings::setCacheStorageMethod($cacheMethod, $cacheSettings);
```

When your script terminates all entries will be cleared from APC, regardless of the cacheTime value, so it cannot be used for persistent storage using this mechanism.

#### PHPExcel\_CachedObjectStorageFactory::cache\_to\_memcache

When using cache\_to\_memcache, cell objects are maintained in memcache<sup>2</sup> with only an index maintained in PHP memory to identify that the cell exists.

By default, PHPEXcel looks for a memcache server on localhost at port 11211. It also sets a memcache timeout limit of 600 seconds. If you are running memcache on a different server or port, then you can change these defaults when you initialise cache\_to\_memcache:

```
$cacheMethod = PHPEXcel_CachedObjectStorageFactory::cache_to_memcache;
$cacheSettings = array( 'memcacheServer' => 'localhost',
                       'memcachePort'   => 11211,
                       'cacheTime'      => 600
                       );
PHPExcel_Settings::setCacheStorageMethod($cacheMethod, $cacheSettings);
```

When your script terminates all entries will be cleared from memcache, regardless of the cacheTime value, so it cannot be used for persistent storage using this mechanism.

#### PHPExcel\_CachedObjectStorageFactory::cache\_to\_wincache;

When using cache\_to\_wincache, cell objects are maintained in Wincache<sup>3</sup> with only an index maintained in PHP memory to identify that the cell exists. By default, a Wincache cache timeout of 600 seconds is used, which should be enough for most applications: although it is possible to change this when initialising cache\_to\_wincache.

```
$cacheMethod = PHPEXcel_CachedObjectStorageFactory::cache_to_wincache;
$cacheSettings = array( 'cacheTime'      => 600
                       );
PHPExcel_Settings::setCacheStorageMethod($cacheMethod, $cacheSettings);
```

When your script terminates all entries will be cleared from Wincache, regardless of the cacheTime value, so it cannot be used for persistent storage using this mechanism.

#### PHPExcel\_CachedObjectStorageFactory::cache\_to\_sqlite;

Uses an SQLite 2 in-memory database for caching cell data. Unlike other caching methods, neither cells nor an index are held in PHP memory - an indexed database table makes it unnecessary to hold any index in PHP memory - making this the most memory-efficient of the cell caching methods.

#### PHPExcel\_CachedObjectStorageFactory::cache\_to\_sqlite3;

Uses an SQLite 3 in-memory database for caching cell data. Unlike other caching methods, neither cells nor an index are held in PHP memory - an indexed database table makes it unnecessary to hold any index in PHP memory - making this the most memory-efficient of the cell caching methods.

## 4.2.2. Language/Locale

Some localisation elements have been included in PHPEXcel. You can set a locale by changing the settings. To set the locale to Brazilian Portuguese you would use:

```
$locale = 'pt_br';
$validLocale = PHPEXcel_Settings::setLocale($locale);
if (!$validLocale) {
    echo 'Unable to set locale to '.$locale." - reverting to en_us<br />\n";
}
```

<sup>2</sup> You must have a memcache server running, and have enabled memcache for your PHP to use this option.

<sup>3</sup> You must have Wincache enabled for PHP to use this option.

If Brazilian Portuguese language files aren't available, then the Portuguese will be enabled instead: if Portuguese language files aren't available, then the `setLocale()` method will return an error, and American English (`en_us`) settings will be used throughout.

More details of the features available once a locale has been set, including a list of the languages and locales currently supported, can be found in section 4.6.5 Locale Settings for Formulae.

### 4.3. *Clearing a Workbook from memory*

The PHPEXcel object contains cyclic references (e.g. the workbook is linked to the worksheets, and the worksheets are linked to their parent workbook) which cause problems when PHP tries to clear the objects from memory when they are `unset()`, or at the end of a function when they are in local scope. The result of this is "memory leaks", which can easily use a large amount of PHP's limited memory.

This can only be resolved manually: if you need to unset a workbook, then you also need to "break" these cyclic references before doing so. PHPEXcel provides the `disconnectWorksheets()` method for this purpose.

```
$objPHPExcel->disconnectWorksheets();  
unset($objPHPExcel);
```

### 4.4. *Worksheets*

A worksheet is a collection of cells, formula's, images, graphs, ... It holds all data necessary to represent as a spreadsheet worksheet.

When you load a workbook from a spreadsheet file, it will be loaded with all its existing worksheets (unless you specified that only certain sheets should be loaded). When you load from non-spreadsheet files (such as a CSV or HTML file) or from spreadsheet formats that don't identify worksheets by name (such as SYLK), then a single worksheet called "WorkSheet" will be created containing the data from that file.

When you instantiate a new workbook, PHPEXcel will create it with a single worksheet called "WorkSheet".

The `getSheetCount()` method will tell you the number of worksheets in the workbook; while the `getSheetNames()` method will return a list of all worksheets in the workbook, indexed by the order in which their "tabs" would appear when opened in MS Excel (or other appropriate Spreadsheet program).

Individual worksheets can be accessed by name, or by their index position in the workbook. The index position represents the order that each worksheet "tab" is shown when the workbook is opened in MS Excel (or other appropriate Spreadsheet program). To access a sheet by its index, use the `getSheet()` method.

```
// Get the second sheet in the workbook  
// Note that sheets are indexed from 0  
$objPHPExcel->getSheet(1);
```

If you don't specify a sheet index, then the first worksheet will be returned.

Methods also exist allowing you to reorder the worksheets in the workbook.

To access a sheet by name, use the `getSheetByName()` method, specifying the name of the worksheet that you want to access.

```
// Retrieve the worksheet called 'Worksheet 1'  
$objPHPExcel->getSheetByName('Worksheet 1');
```

Alternatively, one worksheet is always the currently active worksheet, and you can access that directly. The currently active worksheet is the one that will be active when the workbook is opened in MS Excel (or other appropriate Spreadsheet program).

```
// Retrieve the current active worksheet  
$objPHPExcel->getActiveSheet();
```

You can change the currently active sheet by index or by name using the `setActiveSheetIndex()` and `setActiveSheetIndexByName()` methods.

#### 4.4.1. Adding a new Worksheet

You can add a new worksheet to the workbook using the `createSheet()` method of the `PHPExcel` object. By default, this will be created as a new “last” sheet; but you can also specify an index position as an argument, and the worksheet will be inserted at that position, shuffling all subsequent worksheets in the collection down a place.

```
$objPHPExcel->createSheet();
```

A new worksheet created using this method will be called “Worksheet” or “Worksheet<n>” where “<n>” is the lowest number possible to guarantee that the title is unique.

Alternatively, you can instantiate a new worksheet (setting the title to whatever you choose) and then insert it into your workbook using the `addSheet()` method.

```
// Create a new worksheet called "My Data"
$myWorkSheet = new PHPExcel_Worksheet($objPHPExcel, 'My Data');
// Attach the "My Data" worksheet as the first worksheet in the PHPExcel object
$objPHPExcel->addSheet($myWorkSheet, 0);
```

If you don't specify an index position as the second argument, then the new worksheet will be added after the last existing worksheet.

#### 4.4.2. Copying Worksheets

Sheets within the same workbook can be copied by creating a clone of the worksheet you wish to copy, and then using the `addSheet()` method to insert the clone into the workbook.

```
$objClonedWorksheet = clone $objPHPExcel->getSheetByName('Worksheet 1');
$objClonedWorksheet->setTitle('Copy of Worksheet 1')
$objPHPExcel->addSheet($objClonedWorksheet);
```

You can also copy worksheets from one workbook to another, though this is more complex as `PHPExcel` also has to replicate the styling between the two workbooks. The `addExternalSheet()` method is provided for this purpose.

```
$objClonedWorksheet = clone $objPHPExcel1->getSheetByName('Worksheet 1');
$objPHPExcel->addExternalSheet($objClonedWorksheet);
```

In both cases, it is the developer's responsibility to ensure that worksheet names are not duplicated. `PHPExcel` will throw an exception if you attempt to copy worksheets that will result in a duplicate name.

#### 4.4.3. Removing a Worksheet

You can delete a worksheet from a workbook, identified by its index position, using the `removeSheetByIndex()` method

```
$sheetIndex = $objPHPExcel->getIndex($objPHPExcel->getSheetByName('Worksheet 1'));
$objPHPExcel->removeSheetByIndex($sheetIndex);
```

If the currently active worksheet is deleted, then the sheet at the previous index position will become the currently active sheet.

### 4.5. Accessing cells

Accessing cells in a `PHPExcel` worksheet should be pretty straightforward. This topic lists some of the options to access a cell.

#### 4.5.1. Setting a cell value by coordinate

Setting a cell value by coordinate can be done using the worksheet's `setCellValue()` method.

```
$objPHPExcel->getActiveSheet()->setCellValue('B8', 'Some value');
```

### 4.5.2. Retrieving a cell by coordinate

To retrieve the value of a cell, the cell should first be retrieved from the worksheet using the `getCell` method. A cell's value can be read again using the following line of code:

```
$objPHPExcel->getActiveSheet()->getCell('B8')->getValue();
```

If you need the calculated value of a cell, use the following code. This is further explained in 4.6.41.

```
$objPHPExcel->getActiveSheet()->getCell('B8')->getCalculatedValue();
```

### 4.5.3. Setting a cell value by column and row

Setting a cell value by coordinate can be done using the worksheet's `setCellValueByColumnAndRow` method.

```
// Set cell B8
$objPHPExcel->getActiveSheet()->setCellValueByColumnAndRow(1, 8, 'Some value');
```

### 4.5.4. Retrieving a cell by column and row

To retrieve the value of a cell, the cell should first be retrieved from the worksheet using the `getCellByColumnAndRow` method. A cell's value can be read again using the following line of code:

```
// Get cell B8
$objPHPExcel->getActiveSheet()->getCellByColumnAndRow(1, 8)->getValue();
```

If you need the calculated value of a cell, use the following code. This is further explained in 4.6.41

```
// Get cell B8
$objPHPExcel->getActiveSheet()->getCellByColumnAndRow(1, 8)->getCalculatedValue();
```

### 4.5.5. Looping cells

#### Looping cells using iterators

The easiest way to loop cells is by using iterators. Using iterators, one can use `foreach` to loop worksheets, rows and cells.

Below is an example where we read all the values in a worksheet and display them in a table.

```
<?php
$objReader = PHPExcel_IOFactory::createReader('Excel2007');
$objReader->setReadDataOnly(true);

$objPHPExcel = $objReader->load("test.xlsx");
$objWorksheet = $objPHPExcel->getActiveSheet();

echo '<table>' . "\n";
foreach ($objWorksheet->getRowIterator() as $row) {
    echo '<tr>' . "\n";

    $cellIterator = $row->getCellIterator();
    $cellIterator->setIterateOnlyExistingCells(false); // This loops all cells,
                                                    // even if it is not set.
                                                    // By default, only cells
                                                    // that are set will be
                                                    // iterated.

    foreach ($cellIterator as $cell) {
        echo '<td>' . $cell->getValue() . '</td>' . "\n";
    }

    echo '</tr>' . "\n";
}
echo '</table>' . "\n";
?>
```

Note that we have set the cell iterator's `setIterateOnlyExistingCells()` to `false`. This makes the iterator loop all cells, even if they were not set before.

**i** The cell iterator will return `null` as the cell if it is not set in the worksheet. Setting the cell iterator's `setIterateOnlyExistingCells()` to `false` will loop all cells in the worksheet that can be available at that moment. This will create new cells if required and increase memory usage! Only use it if it is intended to loop all cells that are possibly available.

### Looping cells using indexes

One can use the possibility to access cell values by column and row index like (0,1) instead of 'A1' for reading and writing cell values in loops.

**i** Note: In PHPEXcel column index is 0-based while row index is 1-based. That means 'A1' ~ (0,1)

Below is an example where we read all the values in a worksheet and display them in a table.

```
<?php
$objReader = PHPEXcel_IOFactory::createReader('Excel2007');
$objReader->setReadDataOnly(true);

$objPHPExcel = $objReader->load("test.xlsx");
$objWorksheet = $objPHPExcel->getActiveSheet();

$highestRow = $objWorksheet->getHighestRow(); // e.g. 10
$highestColumn = $objWorksheet->getHighestColumn(); // e.g. 'F'

$highestColumnIndex = PHPEXcel_Cell::columnIndexFromString($highestColumn); // e.g. 5

echo '<table>' . "\n";
for ($row = 1; $row <= $highestRow; ++$row) {
    echo '<tr>' . "\n";

    for ($col = 0; $col <= $highestColumnIndex; ++$col) {
        echo '<td>' . $objWorksheet->getCellByColumnAndRow($col, $row)->getValue() .
'</td>' . "\n";
    }

    echo '</tr>' . "\n";
}
echo '</table>' . "\n";
?>
```

### 4.5.6. Using value binders to facilitate data entry

Internally, PHPEXcel uses a default `PHPEXcel_Cell_IValueBinder` implementation (`PHPEXcel_Cell_DefaultValueBinder`) to determine data types of entered data using a cell's `setValue()` method.

Optionally, the default behaviour of PHPEXcel can be modified, allowing easier data entry. For example, a `PHPEXcel_Cell_AdvancedValueBinder` class is present. It automatically converts percentages and dates entered as strings to the correct format, also setting the cell's style information. The following example demonstrates how to set the value binder in PHPEXcel:

```
/** PHPEXcel */
require_once 'PHPExcel.php';

/** PHPEXcel_Cell_AdvancedValueBinder */
require_once 'PHPExcel/Cell/AdvancedValueBinder.php';

/** PHPEXcel_IOFactory */
require_once 'PHPExcel/IOFactory.php';
```

```
// Set value binder
PHPExcel_Cell::setValueBinder( new PHPExcel_Cell_AdvancedValueBinder() );

// Create new PHPExcel object
$objPHPExcel = new PHPExcel();

// ...

// Add some data, resembling some different data types
$objPHPExcel->getActiveSheet()->setCellValue('A4', 'Percentage value:');
$objPHPExcel->getActiveSheet()->setCellValue('B4', '10%');
// Converts to 0.1 and sets percentage cell style
$objPHPExcel->getActiveSheet()->setCellValue('A5', 'Date/time value:');
$objPHPExcel->getActiveSheet()->setCellValue('B5', '21 December 1983');
// Converts to date and sets date format cell style
```

**Creating your own value binder is easy.**

When advanced value binding is required, you can implement the `PHPExcel_Cell_IValueBinder` interface or extend the `PHPExcel_Cell_DefaultValueBinder` or `PHPExcel_Cell_AdvancedValueBinder` classes.

## 4.6. PHPEXcel recipes

The following pages offer you some widely-used PHPEXcel recipes. Please note that these do NOT offer complete documentation on specific PHPEXcel API functions, but just a bump to get you started. If you need specific API functions, please refer to the API documentation.

For example, 4.6.9 Setting a worksheet's page orientation and size covers setting a page orientation to A4. Other paper formats, like US Letter, are not covered in this document, but in the PHPEXcel API documentation.

### 4.6.1. Setting a spreadsheet's metadata

PHPExcel allows an easy way to set a spreadsheet's metadata, using document property accessors. Spreadsheet metadata can be useful for finding a specific document in a file repository or a document management system. For example Microsoft Sharepoint uses document metadata to search for a specific document in its document lists.

Setting spreadsheet metadata is done as follows:

```
$objPHPExcel->getProperties()->setCreator("Maarten Balliauw");
$objPHPExcel->getProperties()->setLastModifiedBy("Maarten Balliauw");
$objPHPExcel->getProperties()->setTitle("Office 2007 XLSX Test Document");
$objPHPExcel->getProperties()->setSubject("Office 2007 XLSX Test Document");
$objPHPExcel->getProperties()->setDescription("Test document for Office 2007 XLSX,
generated using PHP classes.");
$objPHPExcel->getProperties()->setKeywords("office 2007 openxml php");
$objPHPExcel->getProperties()->setCategory("Test result file");
```

### 4.6.2. Setting a spreadsheet's active sheet

The following line of code sets the active sheet index to the first sheet:

```
$objPHPExcel->setActiveSheetIndex(0);
```

### 4.6.3. Write a date or time into a cell

In Excel, dates and Times are stored as numeric values counting the number of days elapsed since 1900-01-01. For example, the date '2008-12-31' is represented as 39813. You can verify this in Microsoft Office Excel by entering that date in a cell and afterwards changing the number format to 'General' so the true numeric value is revealed. Likewise, '3:15 AM' is represented as 0.135417.

PHPExcel works with UST (Universal Standard Time) date and Time values, but does no internal conversions; so it is up to the developer to ensure that values passed to the date/time conversion functions are UST.

Writing a date value in a cell consists of 2 lines of code. Select the method that suits you the best. Here are some examples:

```
/* PHPExcel_Cell_AdvancedValueBinder required for this sample */
require_once 'PHPExcel/Cell/AdvancedValueBinder.php';

// MySQL-like timestamp '2008-12-31' or date string
PHPExcel_Cell::setValueBinder( new PHPExcel_Cell_AdvancedValueBinder() );
$objPHPExcel->getActiveSheet()
    ->setCellValue('D1', '2008-12-31');
$objPHPExcel->getActiveSheet()
    ->getStyle('D1')
    ->getNumberFormat()
    ->setFormatCode(PHPExcel_Style_NumberFormat::FORMAT_DATE_YYYYMMDDSLASH)
```

```
// PHP-time (Unix time)
$time = gmmktime(0,0,0,12,31,2008); // int(1230681600)
$objPHPExcel->getActiveSheet()
    ->setCellValue('D1', PHPExcel_Shared_Date::PHPToExcel($time));
$objPHPExcel->getActiveSheet()
    ->getStyle('D1')
    ->getNumberFormat()
    ->setFormatCode(PHPExcel_Style_NumberFormat::FORMAT_DATE_YYYYMMDDSLASH)
```

```
// Excel-date/time
$objPHPExcel->getActiveSheet()
    ->setCellValue('D1', 39813)
$objPHPExcel->getActiveSheet()
    ->getStyle('D1')
    ->getNumberFormat()
    ->setFormatCode(PHPExcel_Style_NumberFormat::FORMAT_DATE_YYYYMMDDSLASH)
```

The above methods for entering a date all yield the same result. `PHPExcel_Style_NumberFormat` provides a lot of pre-defined date formats.

The `PHPExcel_Shared_Date::PHPToExcel()` method will also work with a PHP `DateTime` object.

Similarly, times (or date and time values) can be entered in the same fashion: just remember to use an appropriate format code.

#### Notes:

1. See section "Using value binders to facilitate data entry" to learn more about the `AdvancedValueBinder` used in the first example.
2. In previous versions of `PHPExcel` up to and including 1.6.6, when a cell had a date-like number format code, it was possible to enter a date directly using an integer `PHP-time` without converting to Excel date format. Starting with `PHPExcel` 1.6.7 this is no longer supported.
3. Excel can also operate in a 1904-based calendar (default for workbooks saved on Mac). Normally, you do not have to worry about this when using `PHPExcel`.

#### 4.6.4. Write a formula into a cell

Inside the Excel file, formulas are always stored as they would appear in an English version of Microsoft Office Excel, and `PHPExcel` handles all formulae internally in this format. This means that the following rules hold:

- Decimal separator is '.' (period)
- Function argument separator is ',' (comma)
- Matrix row separator is ';' (semicolon)

- English function names must be used

This is regardless of which language version of Microsoft Office Excel may have been used to create the Excel file.

When the final workbook is opened by the user, Microsoft Office Excel will take care of displaying the formula according to the applications language. Translation is taken care of by the application!

The following line of code writes the formula “=IF(C4>500,"profit","loss”) into the cell B8. Note that the formula must start with “=” to make PHPExcel recognise this as a formula.

```
$objPHPExcel->getActiveSheet()->setCellValue('B8','=IF(C4>500,"profit","loss")');
```

If you want to write a string beginning with an “=” to a cell, then you should use the `setCellValueExplicit()` method.

```
$objPHPExcel->getActiveSheet()
    ->setCellValueExplicit('B8',
        '=IF(C4>500,"profit","loss")',
        PHPExcel_Cell_DataType::TYPE_STRING
    );
```

A cell’s formula can be read again using the following line of code:

```
$formula = $objPHPExcel->getActiveSheet()->getCell('B8')->getValue();
```

If you need the calculated value of a cell, use the following code. This is further explained in 4.6.41.

```
$value = $objPHPExcel->getActiveSheet()->getCell('B8')->getCalculatedValue();
```

## 4.6.5. Locale Settings for Formulae

Some localisation elements have been included in PHPExcel. You can set a locale by changing the settings. To set the locale to Russian you would use:

```
$locale = 'ru';
$validLocale = PHPExcel_Settings::setLocale($locale);
if (!$validLocale) {
    echo 'Unable to set locale to '.$locale." - reverting to en_us<br />\n";
}
```

If Russian language files aren’t available, the `setLocale()` method will return an error, and English settings will be used throughout.

Once you have set a locale, you can translate a formula from its internal English coding.

```
$formula = $objPHPExcel->getActiveSheet()->getCell('B8')->getValue();
$translatedFormula =
    PHPExcel_Calculation::getInstance()->_translateFormulaToLocale($formula);
```

You can also create a formula using the function names and argument separators appropriate to the defined locale; then translate it to English before setting the cell value:

```
$formula = '=ДНЕЙ360(ДАТА(2010;2;5);ДАТА(2010;12;31);ИСТИНА)';
$internalFormula =
    PHPExcel_Calculation::getInstance()->translateFormulaToEnglish($formula);
$objPHPExcel->getActiveSheet()->setCellValue('B8',$internalFormula);
```

Currently, formula translation only translates the function names, the constants TRUE and FALSE, and the function argument separators.

At present, the following locale settings are supported:

Language		Locale Code
Czech	Čeština	cs
Danish	Dansk	da
German	Deutsch	de

Language		Locale Code
Spanish	Español	es
Finnish	Suomi	fi
French	Français	fr
Hungarian	Magyar	hu
Italian	Italiano	it
Dutch	Nederlands	nl
Norwegian	Norsk	no
Polish	Język polski	pl
Portuguese	Português	pt
Brazilian Portuguese	Português Brasileiro	pt_br
Russian	русский язык	ru
Swedish	Svenska	sv
Turkish	Türkçe	tr

#### 4.6.6. Write a newline character "\n" in a cell (ALT+"Enter")

In Microsoft Office Excel you get a line break in a cell by hitting ALT+"Enter". When you do that, it automatically turns on "wrap text" for the cell.

Here is how to achieve this in PHPEXcel:

```
$objPHPExcel->getActiveSheet()->getCell('A1')->setValue("hello\nworld");
$objPHPExcel->getActiveSheet()->getStyle('A1')->getAlignment()->setWrapText(true);
```



#### Tip

Read more about formatting cells using `getStyle()` elsewhere.



#### Tip

`AdvancedValueBinder.php` automatically turns on "wrap text" for the cell when it sees a newline character in a string that you are inserting in a cell. Just like Microsoft Office Excel. Try this:

```
require_once 'PHPExcel/Cell/AdvancedValueBinder.php';
PHPExcel_Cell::setValueBinder( new PHPExcel_Cell_AdvancedValueBinder() );

$objPHPExcel->getActiveSheet()->getCell('A1')->setValue("hello\nworld");
```

Read more about `AdvancedValueBinder.php` elsewhere.

#### 4.6.7. Explicitly set a cell's datatype

You can set a cell's datatype explicitly by using the cell's `setValueExplicit` method, or the `setCellValueExplicit` method of a worksheet. Here's an example:

```
$objPHPExcel->getActiveSheet()->getCell('A1')->setValueExplicit('25',
PHPExcel_Cell_DataType::TYPE_NUMERIC);
```

#### 4.6.8. Change a cell into a clickable URL

You can make a cell a clickable URL by setting its hyperlink property:

```
$objPHPExcel->getActiveSheet()->setCellValue('E26', 'www.phpexcel.net');
$objPHPExcel->getActiveSheet()->getCell('E26')->getHyperlink()-
>setUrl('http://www.phpexcel.net');
```

If you want to make a hyperlink to another worksheet/cell, use the following code:

```
$objPHPExcel->getActiveSheet()->setCellValue('E26', 'www.phpexcel.net');
$objPHPExcel->getActiveSheet()->getCell('E26')->getHyperlink()-
>setUrl("sheet://'Sheetname'!A1");
```

#### 4.6.9. Setting a worksheet's page orientation and size

Setting a worksheet's page orientation and size can be done using the following lines of code:

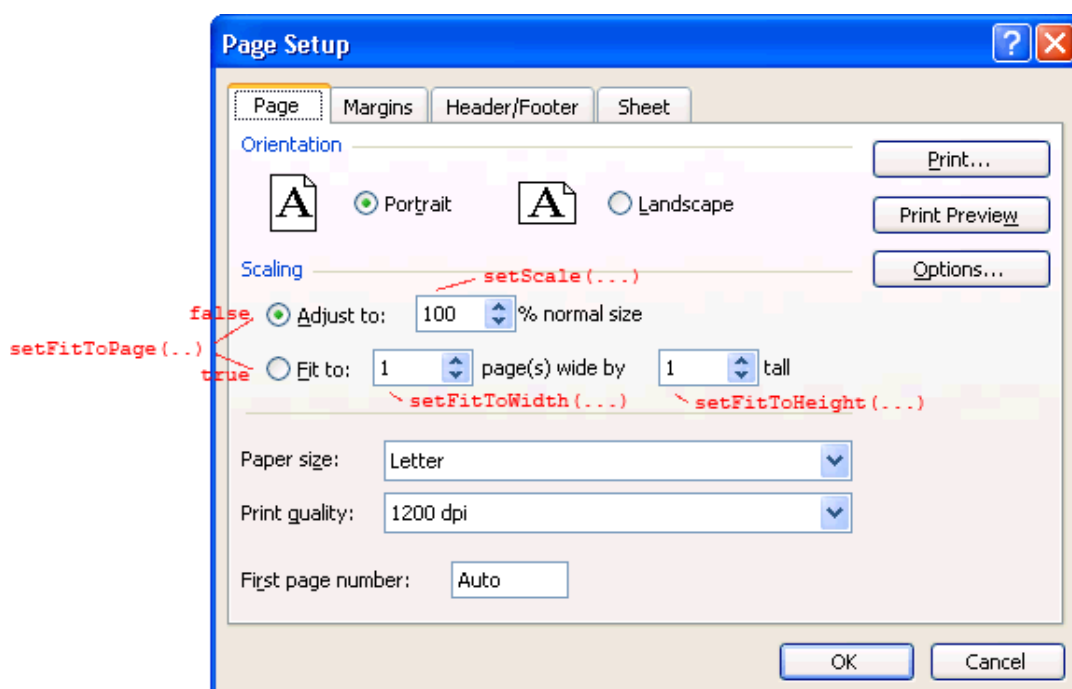
```
$objPHPExcel->getActiveSheet()->getPageSetup()-
>setOrientation(PHPExcel_Worksheet_PageSetup::ORIENTATION_LANDSCAPE);
$objPHPExcel->getActiveSheet()->getPageSetup()-
>setPaperSize(PHPExcel_Worksheet_PageSetup::PAPERSIZE_A4);
```

Note that there are additional page settings available. Please refer to the API documentation for all possible options.

#### 4.6.10. Page Setup: Scaling options

The page setup scaling options in PHPEXcel relate directly to the scaling options in the "Page Setup" dialog as shown in the illustration.

Default values in PHPEXcel correspond to default values in MS Office Excel as shown in illustration



method	initial value	calling method will trigger	Note
setFitToPage(...)	false	-	
setScale(...)	100	setFitToPage(false)	
setFitToWidth(...)	1	setFitToPage(true)	value 0 means do-not-fit-to-width
setFitToHeight(...)	1	setFitToPage(true)	value 0 means do-not-fit-to-height

#### Example

Here is how to fit to 1 page wide by infinite pages tall:

```
$objPHPExcel->getActiveSheet()->getPageSetup()->setFitToWidth(1);
$objPHPExcel->getActiveSheet()->getPageSetup()->setFitToHeight(0);
```

As you can see, it is not necessary to call `setFitToPage(true)` since `setFitToWidth(...)` and `setFitToHeight(...)` triggers this.

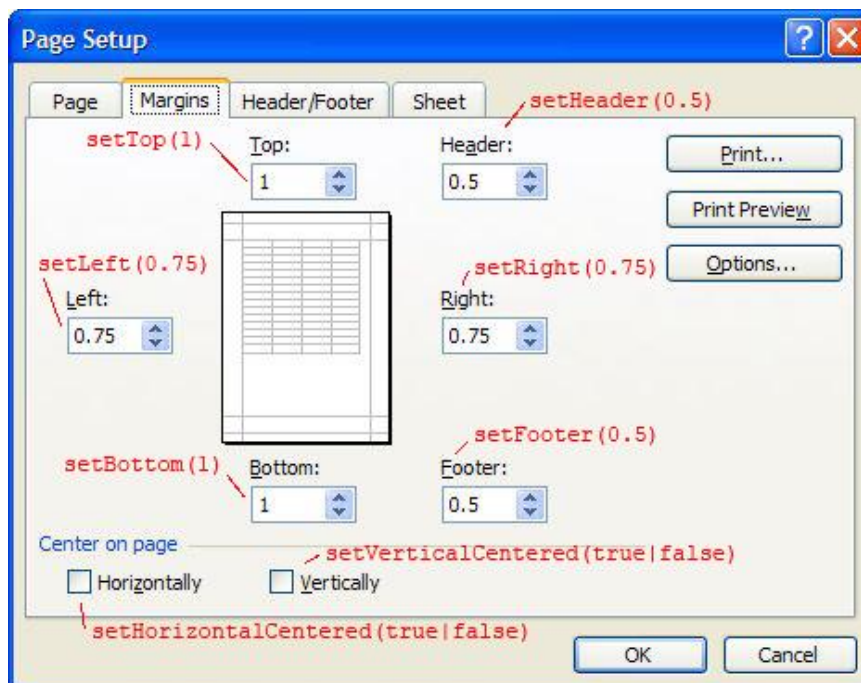
**i** If you use `setFitToWidth()` you should in general also specify `setFitToHeight()` explicitly like in the example. Be careful relying on the initial values. This is especially true if you are upgrading from PHPEXcel 1.7.0 to 1.7.1 where the default values for fit-to-height and fit-to-width changed from 0 to 1.

#### 4.6.11. Page margins

To set page margins for a worksheet, use this code:

```
$objPHPExcel->getActiveSheet()->getPageMargins()->setTop(1);
$objPHPExcel->getActiveSheet()->getPageMargins()->setRight(0.75);
$objPHPExcel->getActiveSheet()->getPageMargins()->setLeft(0.75);
$objPHPExcel->getActiveSheet()->getPageMargins()->setBottom(1);
```

Note that the margin values are specified in inches.



#### 4.6.12. Center a page horizontally/vertically

To center a page horizontally/vertically, you can use the following code:

```
$objPHPExcel->getActiveSheet()->getPageSetup()->setHorizontalCentered(true);
$objPHPExcel->getActiveSheet()->getPageSetup()->setVerticalCentered(false);
```

#### 4.6.13. Setting the print header and footer of a worksheet

Setting a worksheet's print header and footer can be done using the following lines of code:

```
$objPHPExcel->getActiveSheet()->getHeaderFooter()->setOddHeader('&C&HPlease treat
this document as confidential!');
$objPHPExcel->getActiveSheet()->getHeaderFooter()->setOddFooter('&L&B' .
$objPHPExcel->getProperties()->getTitle() . '&RPage &P of &N');
```

Substitution and formatting codes (starting with `&`) can be used inside headers and footers. There is no required order in which these codes must appear.

The first occurrence of the following codes turns the formatting ON, the second occurrence turns it OFF again:

- » Strikethrough
- » Superscript
- » Subscript

Superscript and subscript cannot both be ON at same time. Whichever comes first wins and the other is ignored, while the first is ON.

The following codes are supported by Excel2007:

<b>&amp;L</b>	Code for "left section" (there are three header / footer locations, "left", "center", and "right"). When two or more occurrences of this section marker exist, the contents from all markers are concatenated, in the order of appearance, and placed into the left section.
<b>&amp;P</b>	Code for "current page #"
<b>&amp;N</b>	Code for "total pages"
<b>&amp;font size</b>	Code for "text font size", where font size is a font size in points.
<b>&amp;K</b>	Code for "text font color" <ul style="list-style-type: none"> <li>» RGB Color is specified as RRGGBB</li> <li>» Theme Color is specifed as TTNN where TT is the theme color Id, S is either "+" or "-" of the tint/shade value, NN is the tint/shade value.</li> </ul>
<b>&amp;S</b>	Code for "text strikethrough" on / off
<b>&amp;X</b>	Code for "text super script" on / off
<b>&amp;Y</b>	Code for "text subscript" on / off
<b>&amp;C</b>	Code for "center section". When two or more occurrences of this section marker exist, the contents from all markers are concatenated, in the order of appearance, and placed into the center section.
<b>&amp;D</b>	Code for "date"
<b>&amp;T</b>	Code for "time"
<b>&amp;G</b>	Code for "picture as background" <p>Please make sure to add the image to the header/footer:</p> <pre>\$objDrawing = new PHPExcel_Worksheet_HeaderFooterDrawing(); \$objDrawing-&gt;setName('PHPExcel logo'); \$objDrawing-&gt;setPath('./images/phpexcel_logo.gif'); \$objDrawing-&gt;setHeight(36); \$objPHPExcel-&gt;getActiveSheet()-&gt;getHeaderFooter()- &gt;addImage(\$objDrawing, PHPExcel_Worksheet_HeaderFooter::IMAGE_HEADER_LEFT);</pre>
<b>&amp;U</b>	Code for "text single underline"
<b>&amp;E</b>	Code for "double underline"
<b>&amp;R</b>	Code for "right section". When two or more occurrences of this section marker exist, the contents from all markers are concatenated, in the order of appearance, and placed into the right section.
<b>&amp;Z</b>	Code for "this workbook's file path"
<b>&amp;F</b>	Code for "this workbook's file name"
<b>&amp;A</b>	Code for "sheet tab name"
<b>&amp;+</b>	Code for add to page #
<b>&amp;-</b>	Code for subtract from page #
<b>&amp;"font name,font type"</b>	Code for "text font name" and "text font type", where font name and font type are strings specifying the name and type of the font, separated by a comma. When a hyphen appears in font name, it means "none specified". Both of font name and font type can be localized values.
<b>&amp;"-,Bold"</b>	Code for "bold font style"
<b>&amp;B</b>	Code for "bold font style"
<b>&amp;"-,Regular"</b>	Code for "regular font style"
<b>&amp;"-,Italic"</b>	Code for "italic font style"
<b>&amp;I</b>	Code for "italic font style"

&"-,Bold Italic"	Code for "bold italic font style"
&O	Code for "outline style"
&H	Code for "shadow style"

**Tip**

The above table of codes may seem overwhelming first time you are trying to figure out how to write some header or footer. Luckily, there is an easier way. Let Microsoft Office Excel do the work for you.

For example, create in Microsoft Office Excel an `xlsx` file where you insert the header and footer as desired using the programs own interface. Save file as `test.xlsx`. Now, take that file and read off the values using PHPEXcel as follows:

```
$objPHPExcel = PHPEXcel_IOFactory::load('test.xlsx');
$objWorksheet = $objPHPExcel->getActiveSheet();
var_dump($objWorksheet->getHeaderFooter()->getOddFooter());
var_dump($objWorksheet->getHeaderFooter()->getEvenFooter());
var_dump($objWorksheet->getHeaderFooter()->getOddHeader());
var_dump($objWorksheet->getHeaderFooter()->getEvenHeader());
```

That reveals the codes for the even/odd header and footer. Experienced users may find it easier to rename `test.xlsx` to `test.zip`, unzip it, and inspect directly the contents of the relevant `xl/worksheets/sheetX.xml` to find the codes for header/footer.

**4.6.14. Setting printing breaks on a row or column**

To set a print break, use the following code, which sets a row break on row 10.

```
$objPHPExcel->getActiveSheet()->setBreak('A10', PHPEXcel_Worksheet::BREAK_ROW);
```

The following line of code sets a print break on column D:

```
$objPHPExcel->getActiveSheet()->setBreak('D10', PHPEXcel_Worksheet::BREAK_COLUMN);
```

**4.6.15. Show/hide gridlines when printing**

To show/hide gridlines when printing, use the following code:

```
$objPHPExcel->getActiveSheet()->setShowGridlines(true);
```

**4.6.16. Setting rows/columns to repeat at top/left**

PHPExcel can repeat specific rows/cells at top/left of a page. The following code is an example of how to repeat row 1 to 5 on each printed page of a specific worksheet:

```
$objPHPExcel->getActiveSheet()->getPageSetup()->setRowsToRepeatAtTopByStartAndEnd(1, 5);
```

**4.6.17. Specify printing area**

To specify a worksheet's printing area, use the following code:

```
$objPHPExcel->getActiveSheet()->getPageSetup()->setPrintArea('A1:E5');
```

There can also be multiple printing areas in a single worksheet:

```
$objPHPExcel->getActiveSheet()->getPageSetup()->setPrintArea('A1:E5,G4:M20');
```

**4.6.18. Formatting cells**

A cell can be formatted with font, border, fill, ... style information. For example, one can set the foreground colour of a cell to red, aligned to the right, and the border to black and thick border style. Let's do that on cell B2:

```
$objPHPExcel->getActiveSheet()->getStyle('B2')->getFont()->getColor()->setARGB(PHPEXcel_Style_Color::COLOR_RED);

$objPHPExcel->getActiveSheet()->getStyle('B2')->getAlignment()->setHorizontal(PHPEXcel_Style_Alignment::HORIZONTAL_RIGHT);
```

```

$objPHPExcel->getActiveSheet()->getStyle('B2')->getBorders()->getTop()-
>setBorderStyle(PHPExcel_Style_Border::BORDER_THICK);
$objPHPExcel->getActiveSheet()->getStyle('B2')->getBorders()->getBottom()-
>setBorderStyle(PHPExcel_Style_Border::BORDER_THICK);
$objPHPExcel->getActiveSheet()->getStyle('B2')->getBorders()->getLeft()-
>setBorderStyle(PHPExcel_Style_Border::BORDER_THICK);
$objPHPExcel->getActiveSheet()->getStyle('B2')->getBorders()->getRight()-
>setBorderStyle(PHPExcel_Style_Border::BORDER_THICK);

$objPHPExcel->getActiveSheet()->getStyle('B2')->getFill()-
>setFillType(PHPExcel_Style_Fill::FILL_SOLID);
$objPHPExcel->getActiveSheet()->getStyle('B2')->getFill()->getStartColor()-
>setARGB('FFFF0000');

```

Starting with PHPExcel 1.7.0 `getStyle()` also accepts a cell range as a parameter. For example, you can set a red background color on a range of cells:

```

$objPHPExcel->getActiveSheet()->getStyle('B3:B7')->getFill()-
->setFillType(PHPExcel_Style_Fill::FILL_SOLID)
->getStartColor()->setARGB('FFFF0000');

```

### Tip

It is recommended to style many cells at once, using e.g. `getStyle('A1:M500')`, rather than styling the cells individually in a loop. This is much faster compared to looping through cells and styling them individually.

There is also an alternative manner to set styles. The following code sets a cell's style to font bold, alignment right, top border thin and a gradient fill:

```

$styleArray = array(
    'font' => array(
        'bold' => true,
    ),
    'alignment' => array(
        'horizontal' => PHPExcel_Style_Alignment::HORIZONTAL_RIGHT,
    ),
    'borders' => array(
        'top' => array(
            'style' => PHPExcel_Style_Border::BORDER_THIN,
        ),
    ),
    'fill' => array(
        'type' => PHPExcel_Style_Fill::FILL_GRADIENT_LINEAR,
        'rotation' => 90,
        'startcolor' => array(
            'argb' => 'FFA0A0A0',
        ),
        'endcolor' => array(
            'argb' => 'FFFFFFF',
        ),
    ),
);

$objPHPExcel->getActiveSheet()->getStyle('A3')->applyFromArray($styleArray);

```

Or with a range of cells:

```

$objPHPExcel->getActiveSheet()->getStyle('B3:B7')->applyFromArray($styleArray);

```

This alternative method using arrays should be faster in terms of execution whenever you are setting more than one style property. But the difference may barely be measurable unless you have many different styles in your workbook.

**i** Prior to PHPExcel 1.7.0 `duplicateStyleArray()` was the recommended method for styling a cell range, but this method has now been deprecated since `getStyle()` has started to accept a cell range.

### 4.6.19. Number formats

You often want to format numbers in Excel. For example you may want a thousands separator plus a fixed number of decimals after the decimal separator. Or perhaps you want some numbers to be zero-padded.

In Microsoft Office Excel you may be familiar with selecting a number format from the "Format Cells" dialog. Here there are some predefined number formats available including some for dates. The dialog is designed in a way so you don't have to interact with the underlying raw number format code unless you need a custom number format.

In PHPExcel, you can also apply various predefined number formats. Example:

```
$objPHPExcel->getActiveSheet()->getStyle('A1')->getNumberFormat()
->setFormatCode(PHPExcel_Style_NumberFormat::FORMAT_NUMBER_COMMA_SEPARATED1);
```

This will format a number e.g. 1587.2 so it shows up as 1,587.20 when you open the workbook in MS Office Excel. (Depending on settings for decimal and thousands separators in Microsoft Office Excel it may show up as 1.587,20)

You can achieve exactly the same as the above by using this:

```
$objPHPExcel->getActiveSheet()->getStyle('A1')->getNumberFormat()
->setFormatCode('#,##0.00');
```

In Microsoft Office Excel, as well as in PHPExcel, you will have to interact with raw number format codes whenever you need some special custom number format. Example:

```
$objPHPExcel->getActiveSheet()->getStyle('A1')->getNumberFormat()
->setFormatCode('[Blue] [>=3000] $#,##0; [Red] [<0] $#,##0; $#,##0');
```

Another example is when you want numbers zero-padded with leading zeros to a fixed length:

```
$objPHPExcel->getActiveSheet()->getCell('A1')->setValue(19);
$objPHPExcel->getActiveSheet()->getStyle('A1')->getNumberFormat()
->setFormatCode('0000'); // will show as 0019 in Excel
```

#### Tip

The rules for composing a number format code in Excel can be rather complicated. Sometimes you know how to create some number format in Microsoft Office Excel, but don't know what the underlying number format code looks like. How do you find it?

The readers shipped with PHPExcel come to the rescue. Load your template workbook using e.g. Excel2007 reader to reveal the number format code. Example how read a number format code for cell A1:

```
$objReader = PHPExcel_IOFactory::createReader('Excel2007');
$objPHPExcel = $objReader->load('template.xlsx');
var_dump($objPHPExcel->getActiveSheet()->getStyle('A1')->getNumberFormat()
->getFormatCode());
```

Advanced users may find it faster to inspect the number format code directly by renaming `template.xlsx` to `template.zip`, unzipping, and looking for the relevant piece of XML code holding the number format code in `xl/styles.xml`.

### 4.6.20. Alignment and wrap text

Let's set vertical alignment to the top for cells A1:D4

```
$objPHPExcel->getActiveSheet()->getStyle('A1:D4')
->getAlignment()->setVertical(PHPExcel_Style_Alignment::VERTICAL_TOP);
```

Here is how to achieve wrap text:

```
$objPHPExcel->getActiveSheet()->getStyle('A1:D4')
->getAlignment()->setWrapText(true);
```

#### 4.6.21. Setting the default style of a workbook

It is possible to set the default style of a workbook. Let's set the default font to Arial size 8:

```
$objPHPExcel->getDefaultStyle()->getFont()->setName('Arial');
$objPHPExcel->getDefaultStyle()->getFont()->setSize(8);
```

#### 4.6.22. Styling cell borders

In PHPExcel it is easy to apply various borders on a rectangular selection. Here is how to apply a thick red border outline around cells B2:G8.

```
$styleArray = array(
    'borders' => array(
        'outline' => array(
            'style' => PHPExcel_Style_Border::BORDER_THICK,
            'color' => array('argb' => 'FFFF0000'),
        ),
    ),
);
$objWorksheet->getStyle('B2:G8')->applyFromArray($styleArray);
```

In Microsoft Office Excel, the above operation would correspond to selecting the cells B2:G8, launching the style dialog, choosing a thick red border, and clicking on the "Outline" border component.

**i** Note that the border outline is applied to the rectangular selection B2:G8 as a whole, not on each cell individually.

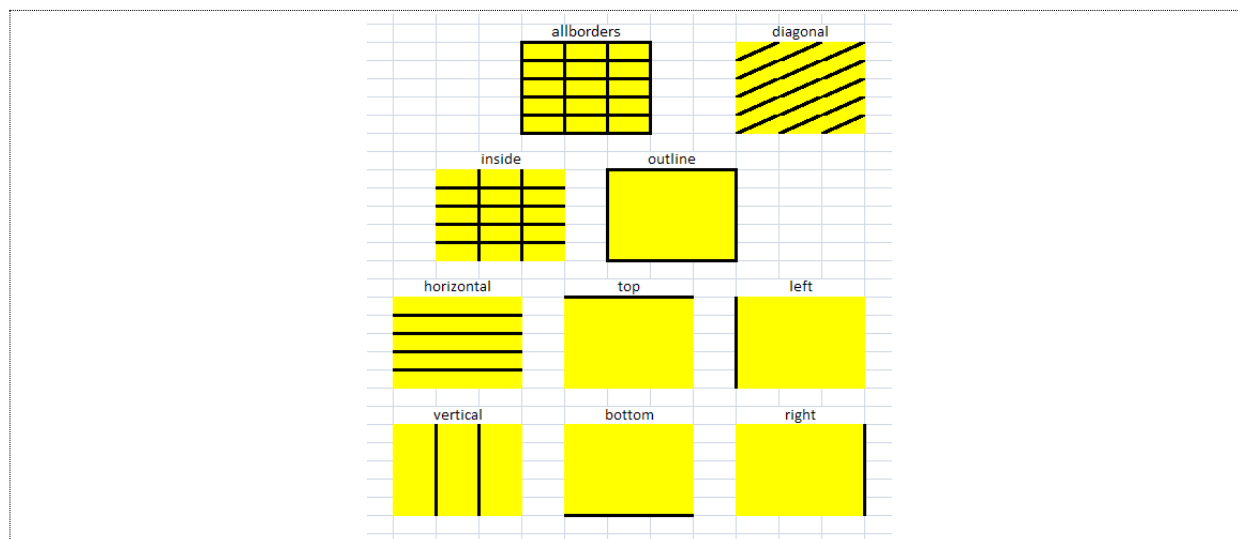
You can achieve any border effect by using just the 5 basic borders and operating on a single cell at a time:

Array key	Maps to property
left	getLeft()
right	getRight()
top	getTop()
bottom	getBottom()
diagonal	getDiagonal()

Additional shortcut borders come in handy like in the example above. These are the shortcut borders available:

Array key	Maps to property
allborders	getAllBorders()
outline	getOutline()
inside	getInside()
vertical	getVertical()
horizontal	getHorizontal()

An overview of all border shortcuts can be seen in the following image:



**i** If you simultaneously set e.g. allborders and vertical, then we have "overlapping" borders, and one of the components has to win over the other where there is border overlap. In PHPExcel, from weakest to strongest borders, the list is as follows: allborders, outline/inside, vertical/horizontal, left/right/top/bottom/diagonal.

This border hierarchy can be utilized to achieve various effects in an easy manner.

#### 4.6.23. Conditional formatting a cell

A cell can be formatted conditionally, based on a specific rule. For example, one can set the foreground colour of a cell to red if its value is below zero, and to green if its value is zero or more.

One can set a conditional style ruleset to a cell using the following code:

```
$objConditional1 = new PHPExcel_Style_Conditional();
$objConditional1->setConditionType(PHPExcel_Style_Conditional::CONDITION_CELLIS);
$objConditional1->setOperatorType(PHPExcel_Style_Conditional::OPERATOR_LESSTHAN);
$objConditional1->addCondition('0');
$objConditional1->getStyle()->getFont()->getColor()-
>setARGB(PHPExcel_Style_Color::COLOR_RED);
$objConditional1->getStyle()->getFont()->setBold(true);

$objConditional2 = new PHPExcel_Style_Conditional();
$objConditional2->setConditionType(PHPExcel_Style_Conditional::CONDITION_CELLIS);
$objConditional2->
>setOperatorType(PHPExcel_Style_Conditional::OPERATOR_GREATERTHANOREQUAL);
$objConditional2->addCondition('0');
$objConditional2->getStyle()->getFont()->getColor()-
>setARGB(PHPExcel_Style_Color::COLOR_GREEN);
$objConditional2->getStyle()->getFont()->setBold(true);

$conditionalStyles = $objPHPExcel->getActiveSheet()->getStyle('B2')-
>getConditionalStyles();
array_push($conditionalStyles, $objConditional1);
array_push($conditionalStyles, $objConditional2);
$objPHPExcel->getActiveSheet()->getStyle('B2')-
>setConditionalStyles($conditionalStyles);
```

If you want to copy the ruleset to other cells, you can duplicate the style object:

```
$objPHPExcel->getActiveSheet()->duplicateStyle($objPHPExcel->getActiveSheet()-
>getStyle('B2'), 'B3:B7');
```

#### 4.6.24. Add a comment to a cell

To add a comment to a cell, use the following code. The example below adds a comment to cell E11:

```

$objPHPExcel->getActiveSheet()->getComment('E11')->setAuthor('PHPExcel');
$objCommentRichText = $objPHPExcel->getActiveSheet()->getComment('E11')->getText()->createTextRun('PHPExcel:');

$objCommentRichText->getFont()->setBold(true);

$objPHPExcel->getActiveSheet()->getComment('E11')->getText()->createTextRun("\r\n");

$objPHPExcel->getActiveSheet()->getComment('E11')->getText()->createTextRun('Total amount on the current invoice, excluding VAT.');
```

	A	B	C	D	E	F	G	H	I
1		Invoice			2007-12-24 #12566				
9					€0,00				
11				Total excl.:	€64,00				
12				VAT:	€13,44				
13				Total incl.:	€77,44				
14									
15									
16									
17									
18									

#### 4.6.25. Apply autofilter to a range of cells

To apply an autofilter to a range of cells, use the following code:

```
$objPHPExcel->getActiveSheet()->setAutoFilter('A1:C9');
```



**Make sure that you always include the complete filter range!**  
Excel does support setting only the caption row, but that's not a best practice...

#### 4.6.26. Setting security on a spreadsheet

Excel offers 3 levels of “protection”: document security, sheet security and cell security.

- Document security allows you to set a password on a complete spreadsheet, allowing changes to be made only when that password is entered.
- Worksheet security offers other security options: you can disallow inserting rows on a specific sheet, disallow sorting, ...
- Cell security offers the option to lock/unlock a cell as well as show/hide the internal formula

An example on setting document security:

```

$objPHPExcel->getSecurity()->setLockWindows(true);
$objPHPExcel->getSecurity()->setLockStructure(true);
$objPHPExcel->getSecurity()->setWorkbookPassword("PHPExcel");
```

An example on setting worksheet security:

```

$objPHPExcel->getActiveSheet()->getProtection()->setPassword('PHPExcel');
$objPHPExcel->getActiveSheet()->getProtection()->setSheet(true);
$objPHPExcel->getActiveSheet()->getProtection()->setSort(true);
$objPHPExcel->getActiveSheet()->getProtection()->setInsertRows(true);
$objPHPExcel->getActiveSheet()->getProtection()->setFormatCells(true);
```

An example on setting cell security:

```

$objPHPExcel->getActiveSheet()->getStyle('B1')->getProtection()->setLocked(
    PHPExcel_Style_Protection::PROTECTION_UNPROTECTED
);
```

**i** Make sure you enable worksheet protection if you need any of the worksheet protection features! This can be done using the following code: `$objPHPExcel->getActiveSheet()->getProtection()->setSheet(true);`

#### 4.6.27. Setting data validation on a cell

Data validation is a powerful feature of Excel2007. It allows to specify an input filter on the data that can be inserted in a specific cell. This filter can be a range (i.e. value must be between 0 and 10), a list (i.e. value must be picked from a list), ...

The following piece of code only allows numbers between 10 and 20 to be entered in cell B3:

```
$objValidation = $objPHPExcel->getActiveSheet()->getCell('B3')
->getDataValidation();
$objValidation->setType( PHPEXcel_Cell_DataValidation::TYPE_WHOLE );
$objValidation->setErrorStyle( PHPEXcel_Cell_DataValidation::STYLE_STOP );
$objValidation->setAllowBlank(true);
$objValidation->setShowInputMessage(true);
$objValidation->setShowErrorMessage(true);
$objValidation->setErrorTitle('Input error');
$objValidation->setError('Number is not allowed!');
$objValidation->setPromptTitle('Allowed input');
$objValidation->setPrompt('Only numbers between 10 and 20 are allowed.');
```

The following piece of code only allows an item picked from a list of data to be entered in cell B3:

```
$objValidation = $objPHPExcel->getActiveSheet()->getCell('B5')
->getDataValidation();
$objValidation->setType( PHPEXcel_Cell_DataValidation::TYPE_LIST );
$objValidation->setErrorStyle( PHPEXcel_Cell_DataValidation::STYLE_INFORMATION );
$objValidation->setAllowBlank(false);
$objValidation->setShowInputMessage(true);
$objValidation->setShowErrorMessage(true);
$objValidation->setShowDropDown(true);
$objValidation->setErrorTitle('Input error');
$objValidation->setError('Value is not in list.');
```

**i** When using a data validation list like above, make sure you put the list between " and " and that you split the items with a comma (,).

**i** It is important to remember that any string participating in an Excel formula is allowed to be maximum 255 characters (not bytes). This sets a limit on how many items you can have in the string "Item A,Item B,Item C". Therefore it is normally a better idea to type the item values directly in some cell range, say A1:A3, and instead use, say, `$objValidation->setFormula1('Sheet!$A$1:$A$3');`. Another benefit is that the item values themselves can contain the comma ',' character itself.

If you need data validation on multiple cells, one can clone the ruleset:

```
$objPHPExcel->getActiveSheet()->getCell('B8')->setDataValidation(clone
$objValidation);
```

#### 4.6.28. Setting a column's width

A column's width can be set using the following code:

```
$objPHPExcel->getActiveSheet()->getColumnDimension('D')->setWidth(12);
```

If you want PHPEXcel to perform an automatic width calculation, use the following code. PHPEXcel will approximate the column with to the width of the widest column value.

```
$objPHPExcel->getActiveSheet()->getColumnDimension('B')->setAutoSize(true);
```

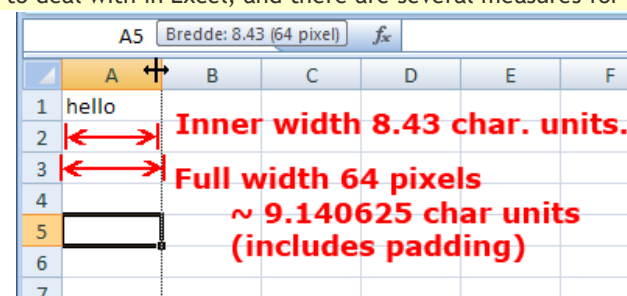
The measure for column width in PHPExcel does not correspond exactly to the measure you may be used to in Microsoft Office Excel. Column widths are difficult to deal with in Excel, and there are several measures for the column width.

- 1) Inner width in character units (e.g. 8.43 this is probably what you are familiar with in Excel)
- 2) Full width in pixels (e.g. 64 pixels)
- 3) Full width in character units (e.g. 9.140625, value -1 indicates unset width)

PHPExcel always operates with 3) "Full width in character units" which is in fact the only value that is stored in any Excel file, hence the most reliable measure. Unfortunately, Microsoft Office Excel does not present you with this measure. Instead measures 1) and 2) are computed by the application when the file is opened and these values are presented in various dialogues and tool tips.

The character width unit is the width of a '0' (zero) glyph in the workbooks default font. Therefore column widths measured in character units in two different workbooks can only be compared if they have the same default workbook font.

If you have some Excel file and need to know the column widths in measure 3), you can read the Excel file with PHPExcel and echo the retrieved values.



#### 4.6.29. Show/hide a column

To set a worksheet's column visibility, you can use the following code. The first line explicitly shows the column C, the second line hides column D.

```
$objPHPExcel->getActiveSheet()->getColumnDimension('C')->setVisible(true);
$objPHPExcel->getActiveSheet()->getColumnDimension('D')->setVisible(false);
```

#### 4.6.30. Group/outline a column

To group/outline a column, you can use the following code:

```
$objPHPExcel->getActiveSheet()->getColumnDimension('E')->setOutlineLevel(1);
```

You can also collapse the column. Note that you should also set the column invisible, otherwise the collapse will not be visible in Excel 2007.

```
$objPHPExcel->getActiveSheet()->getColumnDimension('E')->setCollapsed(true);
$objPHPExcel->getActiveSheet()->getColumnDimension('E')->setVisible(false);
```

Please refer to the part "group/outline a row" for a complete example on collapsing.

You can instruct PHPExcel to add a summary to the right (default), or to the left. The following code adds the summary to the left:

```
$objPHPExcel->getActiveSheet()->setShowSummaryRight(false);
```

#### 4.6.31. Setting a row's height

A row's height can be set using the following code:

```
$objPHPExcel->getActiveSheet()->getRowDimension('10')->setRowHeight(100);
```

#### 4.6.32. Show/hide a row

To set a worksheet's row visibility, you can use the following code. The following example hides row number 10.

```
$objPHPExcel->getActiveSheet()->getRowDimension('10')->setVisible(false);
```

Note that if you apply active filters using an `AutoFilter`, then this will override any rows that you hide or unhide manually within that `AutoFilter` range if you save the file.

#### 4.6.33. Group/outline a row

To group/outline a row, you can use the following code:

```
$objPHPExcel->getActiveSheet()->getRowDimension('5')->setOutlineLevel(1);
```

You can also collapse the row. Note that you should also set the row invisible, otherwise the collapse will not be visible in Excel 2007.

```
$objPHPExcel->getActiveSheet()->getRowDimension('5')->setCollapsed(true);
$objPHPExcel->getActiveSheet()->getRowDimension('5')->setVisible(false);
```

Here's an example which collapses rows 50 to 80:

```
for ($i = 51; $i <= 80; $i++) {
    $objPHPExcel->getActiveSheet()->setCellValue('A' . $i, "FName $i");
    $objPHPExcel->getActiveSheet()->setCellValue('B' . $i, "LName $i");
    $objPHPExcel->getActiveSheet()->setCellValue('C' . $i, "PhoneNo $i");
    $objPHPExcel->getActiveSheet()->setCellValue('D' . $i, "FaxNo $i");
    $objPHPExcel->getActiveSheet()->setCellValue('E' . $i, true);

    $objPHPExcel->getActiveSheet()->getRowDimension($i)->setOutlineLevel(1);
    $objPHPExcel->getActiveSheet()->getRowDimension($i)->setVisible(false);
}
$objPHPExcel->getActiveSheet()->getRowDimension(81)->setCollapsed(true);
```

You can instruct PHPEXcel to add a summary below the collapsible rows (default), or above. The following code adds the summary above:

```
$objPHPExcel->getActiveSheet()->setShowSummaryBelow(false);
```

#### 4.6.34. Merge/unmerge cells

If you have a big piece of data you want to display in a worksheet, you can merge two or more cells together, to become one cell. This can be done using the following code:

```
$objPHPExcel->getActiveSheet()->mergeCells('A18:E22');
```

Removing a merge can be done using the `unmergeCells` method:

```
$objPHPExcel->getActiveSheet()->unmergeCells('A18:E22');
```

#### 4.6.35. Inserting rows/columns

You can insert/remove rows/columns at a specific position. The following code inserts 2 new rows, right before row 7:

```
$objPHPExcel->getActiveSheet()->insertNewRowBefore(7, 2);
```

#### 4.6.36. Add a drawing to a worksheet

A drawing is always represented as a separate object, which can be added to a worksheet.

Therefore, you must first instantiate a new `PHPExcel_Worksheet_Drawing`, and assign its properties a meaningful value:

```
$objDrawing = new PHPEXcel_Worksheet_Drawing();
$objDrawing->setName('Logo');
$objDrawing->setDescription('Logo');
$objDrawing->setPath('./images/officelogo.jpg');
$objDrawing->setHeight(36);
```

To add the above drawing to the worksheet, use the following snippet of code. PHPEXcel creates the link between the drawing and the worksheet:

```
$objDrawing->setWorksheet($objPHPExcel->getActiveSheet());
```

You can set numerous properties on a drawing, here are some examples:

```
$objDrawing->setName('Paid');
```

```

$objDrawing->setDescription('Paid');
$objDrawing->setPath('./images/paid.png');
$objDrawing->setCoordinates('B15');
$objDrawing->setOffsetX(110);
$objDrawing->setRotation(25);
$objDrawing->getShadow()->setVisible(true);
$objDrawing->getShadow()->setDirection(45);

```

You can also add images created using GD functions without needing to save them to disk first as In-Memory drawings.

```

// Use GD to create an in-memory image
$gdImage = @imagecreatetruecolor(120, 20) or die('Cannot Initialize new GD image stream');
$textColor = imagecolorallocate($gdImage, 255, 255, 255);
imagestring($gdImage, 1, 5, 5, 'Created with PHPExcel', $textColor);

// Add the In-Memory image to a worksheet
$objDrawing = new PHPExcel_Worksheet_MemoryDrawing();
$objDrawing->setName('In-Memory image 1');
$objDrawing->setDescription('In-Memory image 1');
$objDrawing->setCoordinates('A1');
$objDrawing->setImageResource($gdImage);
$objDrawing->setRenderingFunction(
    PHPExcel_Worksheet_MemoryDrawing::RENDERING_JPEG
);
$objDrawing->setMimeType(PHPExcel_Worksheet_MemoryDrawing::MIMETYPE_DEFAULT);
$objDrawing->setHeight(36);
$objDrawing->setWorksheet($objPHPExcel->getActiveSheet());

```

#### 4.6.37. Reading Images from a worksheet

A commonly asked question is how to retrieve the images from a workbook that has been loaded, and save them as individual image files to disk.

The following code extracts images from the current active worksheet, and writes each as a separate file.

```

$i = 0;
foreach ($objPHPExcel->getActiveSheet()->getDrawingCollection() as $drawing) {
    if ($drawing instanceof PHPExcel_Worksheet_MemoryDrawing) {
        ob_start();
        call_user_func(
            $drawing->getRenderingFunction(),
            $drawing->getImageResource()
        );
        $imageContents = ob_get_contents();
        ob_end_clean();
        switch ($drawing->getMimeType()) {
            case PHPExcel_Worksheet_MemoryDrawing::MIMETYPE_PNG :
                $extension = 'png'; break;
            case PHPExcel_Worksheet_MemoryDrawing::MIMETYPE_GIF:
                $extension = 'gif'; break;
            case PHPExcel_Worksheet_MemoryDrawing::MIMETYPE_JPEG :
                $extension = 'jpg'; break;
        }
    } else {
        $zipReader = fopen($drawing->getPath(), 'r');
        $imageContents = '';
        while (!feof($zipReader)) {
            $imageContents .= fread($zipReader, 1024);
        }
        fclose($zipReader);
        $extension = $drawing->getExtension();
    }
    $myFileName = '00_Image_' . ++$i . '.' . $extension;
}

```

```
file_put_contents($myFileName,$imageContents);
}
```

#### 4.6.38. Add rich text to a cell

Adding rich text to a cell can be done using `PHPExcel_RichText` instances. Here's an example, which creates the following rich text string:

This invoice is *payable within thirty days after the end of the month* unless specified otherwise on the invoice.

```
$objRichText = new PHPExcel_RichText();
$objRichText->createText('This invoice is ');

$objPayable = $objRichText->createTextRun('payable within thirty days after the end
of the month');
$objPayable->getFont()->setBold(true);
$objPayable->getFont()->setItalic(true);
$objPayable->getFont()->setColor( new PHPExcel_Style_Color(
PHPExcel_Style_Color::COLOR_DARKGREEN ) );

$objRichText->createText(', unless specified otherwise on the invoice.');
```

```
$objPHPExcel->getActiveSheet()->getCell('A18')->setValue($objRichText);
```

#### 4.6.39. Define a named range

PHPExcel supports the definition of named ranges. These can be defined using the following code:

```
// Add some data
$objPHPExcel->setActiveSheetIndex(0);
$objPHPExcel->getActiveSheet()->setCellValue('A1', 'Firstname:');
$objPHPExcel->getActiveSheet()->setCellValue('A2', 'Lastname:');
$objPHPExcel->getActiveSheet()->setCellValue('B1', 'Maarten');
$objPHPExcel->getActiveSheet()->setCellValue('B2', 'Balliauw');

// Define named ranges
$objPHPExcel->addNamedRange( new PHPExcel_NamedRange('PersonFN', $objPHPExcel-
->getActiveSheet(), 'B1') );
$objPHPExcel->addNamedRange( new PHPExcel_NamedRange('PersonLN', $objPHPExcel-
->getActiveSheet(), 'B2') );
```

Optionally, a fourth parameter can be passed defining the named range local (i.e. only usable on the current worksheet). Named ranges are global by default.

#### 4.6.40. Redirect output to a client's web browser

Sometimes, one really wants to output a file to a client's browser, especially when creating spreadsheets on-the-fly. There are some easy steps that can be followed to do this:

1. Create your PHPExcel spreadsheet
2. Output HTTP headers for the type of document you wish to output
3. Use the `PHPExcel_Writer_*` of your choice, and save to "php://output"

`PHPExcel_Writer_Excel2007` uses temporary storage when writing to `php://output`. By default, temporary files are stored in the script's working directory. When there is no access, it falls back to the operating system's temporary files location.



#### This may not be safe for unauthorized viewing!

Depending on the configuration of your operating system, temporary storage can be read by anyone using the same temporary storage folder. When confidentiality of your document is needed, it is recommended not to use `php://output`.

## HTTP headers

Example of a script redirecting an Excel 2007 file to the client's browser:

```
<?php
/* Here there will be some code where you create $objPHPExcel */

// redirect output to client browser
header('Content-Type: application/vnd.openxmlformats-officedocument.spreadsheetml.sheet');
header('Content-Disposition: attachment;filename="myfile.xlsx"');
header('Cache-Control: max-age=0');

$objWriter = PHPExcel_IOFactory::createWriter($objPHPExcel, 'Excel2007');
$objWriter->save('php://output');
?>
```

Example of a script redirecting an Excel5 file to the client's browser:

```
<?php
/* Here there will be some code where you create $objPHPExcel */

// redirect output to client browser
header('Content-Type: application/vnd.ms-excel');
header('Content-Disposition: attachment;filename="myfile.xls"');
header('Cache-Control: max-age=0');

$objWriter = PHPExcel_IOFactory::createWriter($objPHPExcel, 'Excel5');
$objWriter->save('php://output');
?>
```

### Caution:

- Make sure not to include any echo statements or output any other contents than the Excel file. There should be no whitespace before the opening <?php tag and at most one line break after the closing ?> tag (which can also be omitted to avoid problems).
- Make sure that your script is saved without a BOM (Byte-order mark). (Because this counts as echoing output)
- Same things apply to all included files

Failing to follow the above guidelines may result in corrupt Excel files arriving at the client browser, and/or that headers cannot be set by PHP (resulting in warning messages).

### 4.6.41. Setting the default column width

Default column width can be set using the following code:

```
$objPHPExcel->getActiveSheet()->getDefaultColumnDimension()->setWidth(12);
```

### 4.6.42. Setting the default row height

Default row height can be set using the following code:

```
$objPHPExcel->getActiveSheet()->getDefaultRowDimension()->setRowHeight(15);
```

### 4.6.43. Add a GD drawing to a worksheet

There might be a situation where you want to generate an in-memory image using GD and add it to a PHPExcel worksheet without first having to save this file to a temporary location.

Here's an example which generates an image in memory and adds it to the active worksheet:

```
// Generate an image
$gdImage = @imagecreatetruecolor(120, 20) or die('Cannot Initialize new GD image stream');
$textColor = imagecolorallocate($gdImage, 255, 255, 255);
imagestring($gdImage, 1, 5, 5, 'Created with PHPExcel', $textColor);
```

```
// Add a drawing to the worksheet
$objDrawing = new PHPExcel_Worksheet_MemoryDrawing();
$objDrawing->setName('Sample image');
$objDrawing->setDescription('Sample image');
$objDrawing->setImageResource($gdImage);
$objDrawing-
>setRenderingFunction(PHPExcel_Worksheet_MemoryDrawing::RENDERING_JPEG);
$objDrawing->setMimeType(PHPExcel_Worksheet_MemoryDrawing::MIMETYPE_DEFAULT);
$objDrawing->setHeight(36);
$objDrawing->setWorksheet($objPHPExcel->getActiveSheet());
```

#### 4.6.44. Setting worksheet zoom level

To set a worksheet's zoom level, the following code can be used:

```
$objPHPExcel->getActiveSheet()->getSheetView()->setZoomScale(75);
```

Note that zoom level should be in range 10 - 400.

#### 4.6.45. Sheet tab color

Sometimes you want to set a color for sheet tab. For example you can have a red sheet tab:

```
$objWorksheet->getTabColor()->setRGB('FF0000');
```

#### 4.6.46. Creating worksheets in a workbook

If you need to create more worksheets in the workbook, here is how:

```
$objWorksheet1 = $objPHPExcel->createSheet();
$objWorksheet1->setTitle('Another sheet');
```

Think of createSheet() as the "Insert sheet" button in Excel. When you hit that button a new sheet is appended to the existing collection of worksheets in the workbook.

#### 4.6.47. Hidden worksheets (Sheet states)

Set a worksheet to be hidden using this code:

```
$objPHPExcel->getActiveSheet()
->setSheetState(PHPExcel_Worksheet::SHEETSTATE_HIDDEN);
```

Sometimes you may even want the worksheet to be "very hidden". The available sheet states are :

```
PHPExcel_Worksheet::SHEETSTATE_VISIBLE
PHPExcel_Worksheet::SHEETSTATE_HIDDEN
PHPExcel_Worksheet::SHEETSTATE_VERYHIDDEN
```

In Excel the sheet state "very hidden" can only be set programmatically, e.g. with Visual Basic Macro. It is not possible to make such a sheet visible via the user interface.

#### 4.6.48. Right-to-left worksheet

Worksheets can be set individually whether column 'A' should start at left or right side. Default is left. Here is how to set columns from right-to-left.

```
// right-to-left worksheet
$objPHPExcel->getActiveSheet()
->setRightToLeft(true);
```

## 5. Performing formula calculations

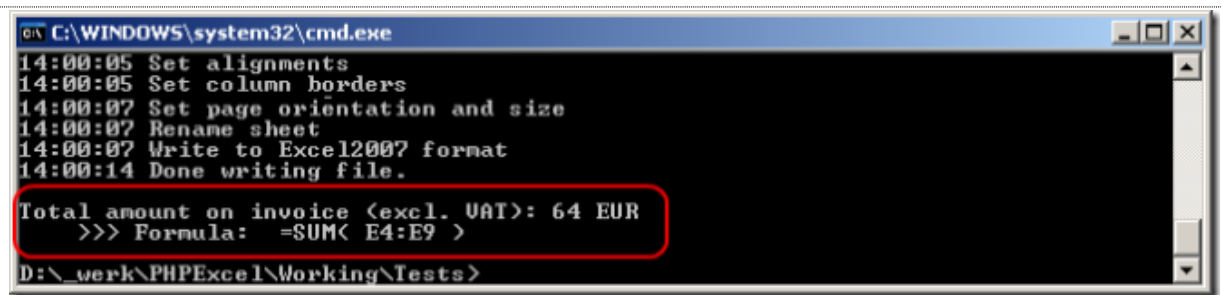
### 5.1. Using the PHPEXcel calculation engine

As PHPEXcel represents an in-memory spreadsheet, it also offers formula calculation capabilities. A cell can be of a value type (containing a number or text), or a formula type (containing a formula which can be evaluated). For example, the formula "`=SUM(A1:A10)`" evaluates to the sum of values in A1, A2, ..., A10.

To calculate a formula, you can call the cell containing the formula's method `getCalculatedValue()`, for example:

```
$objPHPExcel->getActiveSheet()->getCell('E11')->getCalculatedValue();
```

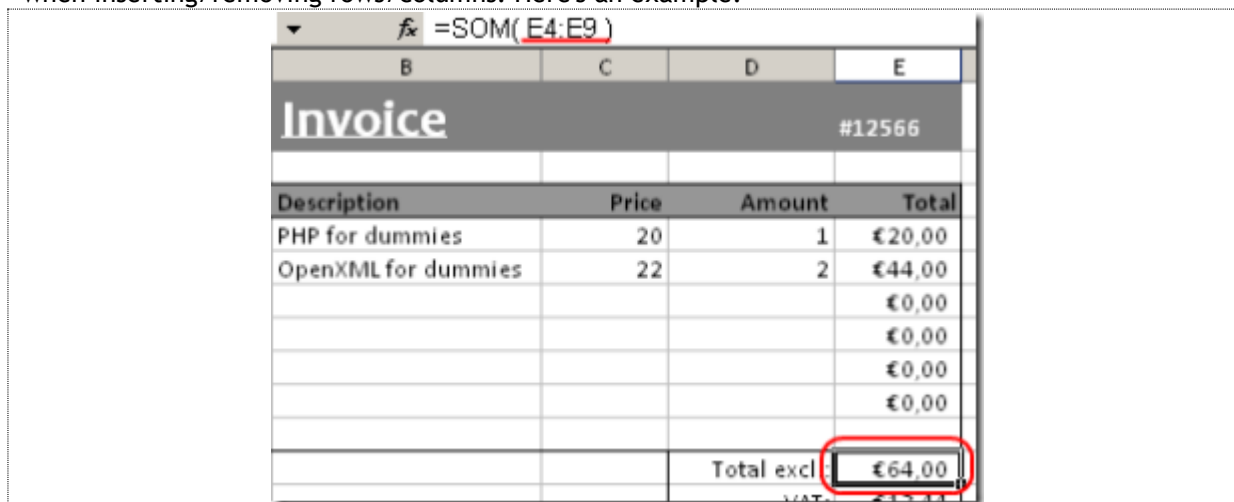
If you write the following line of code in the invoice demo included with PHPEXcel, it evaluates to the value "64":



```

C:\WINDOWS\system32\cmd.exe
14:00:05 Set alignments
14:00:05 Set column borders
14:00:07 Set page orientation and size
14:00:07 Rename sheet
14:00:07 Write to Excel2007 format
14:00:14 Done writing file.
Total amount on invoice (excl. VAT): 64 EUR
>>> Formula: =SUM( E4:E9 )
D:\_werk\PHPExcel\Working\Tests>
  
```

Another nice feature of PHPEXcel's formula parser, is that it can automatically adjust a formula when inserting/removing rows/columns. Here's an example:



	B	C	D	E
	Invoice #12566			
	Description	Price	Amount	Total
	PHP for dummies	20	1	€20,00
	OpenXML for dummies	22	2	€44,00
				€0,00
				€0,00
				€0,00
				€0,00
			Total excl	€64,00

You see that the formula contained in cell E11 is "`SUM(E4:E9)`". Now, when I write the following line of code, two new product lines are added:

```
$objPHPExcel->getActiveSheet()->insertNewRowBefore(7, 2);
```

Did you notice? The formula in the former cell E11 (now E13, as I inserted 2 new rows), changed to "SUM(E4:E11)". Also, the inserted cells duplicate style information of the previous cell, just like Excel's behaviour. Note that you can both insert rows and columns.

There are some known limitations to the PHPExcel calculation engine. Most of them are due to the fact that an Excel formula is converted into PHP code before being executed. This means that Excel formula calculation is subject to PHP's language characteristics.

In Excel '+' wins over '&', just like '\*' wins over '+' in ordinary algebra. The former rule is not what one finds using the calculation engine shipped with PHPExcel.

Reference for operator precedence in PHP:  
<http://www.php.net/operators>

Formulas involving numbers and text may produce unexpected results or even unreadable file contents. For example, the formula `=3+"Hello "` is expected to produce an error in Excel (`#VALUE!`). Due to the fact that PHP converts “Hello” to a numeric value (zero), the result of this formula is evaluated as 3 instead of evaluating as an error. This also causes the Excel document being generated as containing unreadable content.

# PHPExcel Developer Documentation

## 6. Reading and writing to file

As you already know from part 3.4 Readers and writers, reading and writing to a persisted storage is not possible using the base PHPEXcel classes. For this purpose, PHPEXcel provides readers and writers, which are implementations of PHPEXcel\_Reader\_IReader and PHPEXcel\_Writer\_IWriter.

### 6.1. PHPEXcel\_IOFactory

The PHPEXcel API offers multiple methods to create a PHPEXcel\_Reader\_IReader or PHPEXcel\_Writer\_IWriter instance:

- Direct creation
- Via PHPEXcel\_IOFactory

All examples underneath demonstrate the direct creation method. Note that you can also use the PHPEXcel\_IOFactory class to do this.

#### 6.1.1. Creating PHPEXcel\_Reader\_IReader using PHPEXcel\_IOFactory

There are 2 methods for reading in a file into PHPEXcel: using automatic file type resolving or explicitly.

Automatic file type resolving checks the different PHPEXcel\_Reader\_IReader distributed with PHPEXcel. If one of them can load the specified file name, the file is loaded using that PHPEXcel\_Reader\_IReader. Explicit mode requires you to specify which PHPEXcel\_Reader\_IReader should be used.

You can create a PHPEXcel\_Reader\_IReader instance using PHPEXcel\_IOFactory in automatic file type resolving mode using the following code sample:

```
$objPHPExcel = PHPEXcel_IOFactory::load("05featuredemo.xlsx");
```

A typical use of this feature is when you need to read files uploaded by your users, and you don't know whether they are uploading xls or xlsx files.

If you need to set some properties on the reader, (e.g. to only read data, see more about this later), then you may instead want to use this variant:

```
$objReader = PHPEXcel_IOFactory::createReaderForFile("05featuredemo.xlsx");
$objReader->setReadDataOnly(true);
$objReader->load("05featuredemo.xlsx");
```

You can create a PHPEXcel\_Reader\_IReader instance using PHPEXcel\_IOFactory in explicit mode using the following code sample:

```
$objReader = PHPEXcel_IOFactory::createReader("Excel2007");
$objPHPExcel = $objReader->load("05featuredemo.xlsx");
```



Note that automatic type resolving mode is slightly slower than explicit mode.

#### 6.1.2. Creating PHPEXcel\_Writer\_IWriter using PHPEXcel\_IOFactory

You can create a PHPEXcel\_Writer\_IWriter instance using PHPEXcel\_IOFactory:

```
$objWriter = PHPEXcel_IOFactory::createWriter($objPHPExcel, "Excel2007");
$objWriter->save("05featuredemo.xlsx");
```

## 6.2. Excel 2007 (SpreadsheetML) file format

Excel2007 file format is the main file format of PHPEXcel. It allows outputting the in-memory spreadsheet to a .xlsx file.

## 6.2.1. PHPEXcel\_Reader\_Excel2007

### Reading a spreadsheet

You can read an .xlsx file using the following code:

```
$objReader = new PHPEXcel_Reader_Excel2007();  
$objPHPExcel = $objReader->load("05featuredemo.xlsx");
```

### Read data only

You can set the option `setReadDataOnly` on the reader, to instruct the reader to ignore styling, data validation, ... and just read cell data:

```
$objReader = new PHPEXcel_Reader_Excel2007();  
$objReader->setReadDataOnly(true);  
$objPHPExcel = $objReader->load("05featuredemo.xlsx");
```

### Read specific sheets only

You can set the option `setLoadSheetsOnly` on the reader, to instruct the reader to only load the sheets with a given name:

```
$objReader = new PHPEXcel_Reader_Excel2007();  
$objReader->setLoadSheetsOnly( array("Sheet 1", "My special sheet") );  
$objPHPExcel = $objReader->load("05featuredemo.xlsx");
```

### Read specific cells only

You can set the option `setReadFilter` on the reader, to instruct the reader to only load the cells which match a given rule. A read filter can be any class which implements `PHPExcel_Reader_IReadFilter`. By default, all cells are read using the `PHPExcel_Reader_DefaultReadFilter`.

The following code will only read row 1 and rows 20 - 30 of any sheet in the Excel file:

```
class MyReadFilter implements PHPEXcel_Reader_IReadFilter  
{  
    public function readCell($column, $row, $worksheetName = '') {  
        // Read title row and rows 20 - 30  
        if ($row == 1 || ($row >= 20 && $row <= 30)) {  
            return true;  
        }  
        return false;  
    }  
}  
  
$objReader = new PHPEXcel_Reader_Excel2007();  
$objReader->setReadFilter( new MyReadFilter() );  
$objPHPExcel = $objReader->load("06largescale.xlsx");
```

## 6.2.2. PHPEXcel\_Writer\_Excel2007

### Writing a spreadsheet

You can write an .xlsx file using the following code:

```
$objWriter = new PHPEXcel_Writer_Excel2007($objPHPExcel);  
$objWriter->save("05featuredemo.xlsx");
```

### Formula pre-calculation

By default, this writer pre-calculates all formulas in the spreadsheet. This can be slow on large spreadsheets, and maybe even unwanted. You can however disable formula pre-calculation:

```
$objWriter = new PHPEXcel_Writer_Excel2007($objPHPExcel);  
$objWriter->setPreCalculateFormulas(false);  
$objWriter->save("05featuredemo.xlsx");
```

### Office 2003 compatibility pack

Because of a bug in the Office2003 compatibility pack, there can be some small issues when opening Excel2007 spreadsheets (mostly related to formula calculation). You can enable Office2003 compatibility with the following code:

```
$objWriter = new PHPEXcel_Writer_Excel2007($objPHPExcel);
$objWriter->setOffice2003Compatibility(true);
$objWriter->save("05featuredemo.xlsx");
```

**Office2003 compatibility should only be used when needed**

Office2003 compatibility option should only be used when needed. This option disables several Office2007 file format options, resulting in a lower-featured Office2007 spreadsheet when this option is used.

## 6.3. Excel 5 (BIFF) file format

Excel5 file format is the old Excel file format, implemented in PHPEXcel to provide a uniform manner to create both .xlsx and .xls files. It is basically a modified version of [PEAR Spreadsheet\\_Excel\\_Writer](#), although it has been extended and has fewer limitations and more features than the old PEAR library. This can read all BIFF versions that use OLE2: BIFF5 (introduced with office 95) through BIFF8, but cannot read earlier versions.

Excel5 file format will not be developed any further, it just provides an additional file format for PHPEXcel.

**Excel5 (BIFF) limitations**

Please note that BIFF file format has some limits regarding to styling cells and handling large spreadsheets via PHP.

### 6.3.1. PHPEXcel\_Reader\_Excel5

#### Reading a spreadsheet

You can read an .xls file using the following code:

```
$objReader = new PHPEXcel_Reader_Excel5();
$objPHPExcel = $objReader->load("05featuredemo.xls");
```

#### Read data only

You can set the option `setReadDataOnly` on the reader, to instruct the reader to ignore styling, data validation, ... and just read cell data:

```
$objReader = new PHPEXcel_Reader_Excel5();
$objReader->setReadDataOnly(true);
$objPHPExcel = $objReader->load("05featuredemo.xls");
```

#### Read specific sheets only

You can set the option `setLoadSheetsOnly` on the reader, to instruct the reader to only load the sheets with a given name:

```
$objReader = new PHPEXcel_Reader_Excel5();
$objReader->setLoadSheetsOnly( array("Sheet 1", "My special sheet") );
$objPHPExcel = $objReader->load("05featuredemo.xls");
```

#### Read specific cells only

You can set the option `setReadFilter` on the reader, to instruct the reader to only load the cells which match a given rule. A read filter can be any class which implements `PHPExcel_Reader_IReadFilter`. By default, all cells are read using the `PHPExcel_Reader_DefaultReadFilter`.

The following code will only read row 1 and rows 20 - 30 of any sheet in the Excel file:

```
class MyReadFilter implements PHPEXcel_Reader_IReadFilter
{
    public function readCell($column, $row, $worksheetName = '') {
```

```

        // Read title row and rows 20 - 30
        if ($row == 1 || ($row >= 20 && $row <= 30)) {
            return true;
        }

        return false;
    }
}

$objReader = new PHPEXcel_Reader_Excel5();
$objReader->setReadFilter( new MyReadFilter() );
$objPHPExcel = $objReader->load("06largescale.xls");

```

### 6.3.2. PHPEXcel\_Writer\_Excel5

#### Writing a spreadsheet

You can write an .xls file using the following code:

```

$objWriter = new PHPEXcel_Writer_Excel5($objPHPExcel);
$objWriter->save("05featuredemo.xls");

```

## 6.4. Excel 2003 XML file format

Excel 2003 XML file format is a file format which can be used in older versions of Microsoft Excel.



#### Excel 2003 XML limitations

Please note that Excel 2003 XML format has some limits regarding to styling cells and handling large spreadsheets via PHP.

### 6.4.1. PHPEXcel\_Reader\_Excel2003XML

#### Reading a spreadsheet

You can read an .xml file using the following code:

```

$objReader = new PHPEXcel_Reader_Excel2003XML();
$objPHPExcel = $objReader->load("05featuredemo.xml");

```

#### Read specific cells only

You can set the option `setReadFilter` on the reader, to instruct the reader to only load the cells which match a given rule. A read filter can be any class which implements `PHPEXcel_Reader_IReadFilter`. By default, all cells are read using the `PHPEXcel_Reader_DefaultReadFilter`.

The following code will only read row 1 and rows 20 - 30 of any sheet in the Excel file:

```

class MyReadFilter implements PHPEXcel_Reader_IReadFilter
{
    public function readCell($column, $row, $worksheetName = '') {
        // Read title row and rows 20 - 30
        if ($row == 1 || ($row >= 20 && $row <= 30)) {
            return true;
        }

        return false;
    }
}

$objReader = new PHPEXcel_Reader_Excel2003XML();
$objReader->setReadFilter( new MyReadFilter() );
$objPHPExcel = $objReader->load("06largescale.xml");

```

## 6.5. Symbolic Link (SYLK)

Symbolic Link (SYLK) is a Microsoft file format typically used to exchange data between applications, specifically spreadsheets. SYLK files conventionally have a .slk suffix. Composed of only displayable ANSI characters, it can be easily created and processed by other applications, such as databases.

### SYLK limitations

Please note that SYLK file format has some limits regarding to styling cells and handling large spreadsheets via PHP.

### 6.5.1. PHPEXcel\_Reader\_SYLK

#### Reading a spreadsheet

You can read an .slk file using the following code:

```
$objReader = new PHPEXcel_Reader_SYLK();
$objPHPExcel = $objReader->load("05featuredemo.slk");
```

#### Read specific cells only

You can set the option setReadFilter on the reader, to instruct the reader to only load the cells which match a given rule. A read filter can be any class which implements PHPEXcel\_Reader\_IReadFilter. By default, all cells are read using the PHPEXcel\_Reader\_DefaultReadFilter.

The following code will only read row 1 and rows 20 - 30 of any sheet in the SYLK file:

```
class MyReadFilter implements PHPEXcel_Reader_IReadFilter
{
    public function readCell($column, $row, $worksheetName = '') {
        // Read title row and rows 20 - 30
        if ($row == 1 || ($row >= 20 && $row <= 30)) {
            return true;
        }

        return false;
    }
}

$objReader = new PHPEXcel_Reader_SYLK();
$objReader->setReadFilter( new MyReadFilter() );
$objPHPExcel = $objReader->load("06largescale.slk");
```

## 6.6. Open/Libre Office (.ods)

Open Office or Libre Office .ods files are the standard file format for Open Office or Libre Office Calc files.

### 6.6.1. PHPEXcel\_Reader\_OOCalc

#### Reading a spreadsheet

You can read an .ods file using the following code:

```
$objReader = new PHPEXcel_Reader_OOCalc();
$objPHPExcel = $objReader->load("05featuredemo.ods");
```

#### Read specific cells only

You can set the option setReadFilter on the reader, to instruct the reader to only load the cells which match a given rule. A read filter can be any class which implements PHPEXcel\_Reader\_IReadFilter. By default, all cells are read using the PHPEXcel\_Reader\_DefaultReadFilter.

The following code will only read row 1 and rows 20 - 30 of any sheet in the Calc file:

```

class MyReadFilter implements PHPEXcel_Reader_IReadFilter
{
    public function readCell($column, $row, $worksheetName = '') {
        // Read title row and rows 20 - 30
        if ($row == 1 || ($row >= 20 && $row <= 30)) {
            return true;
        }

        return false;
    }
}

$objReader = new PHPEXcel_Reader_OOcalc();
$objReader->setReadFilter( new MyReadFilter() );
$objPHPExcel = $objReader->load("06largescale.ods");

```

## 6.7. CSV (Comma Separated Values)

CSV (Comma Separated Values) are often used as an import/export file format with other systems. PHPEXcel allows reading and writing to CSV files.



### CSV limitations

Please note that CSV file format has some limits regarding to styling cells, number formatting, ...

### 6.7.1. PHPEXcel\_Reader\_CSV

#### Reading a CSV file

You can read a .csv file using the following code:

```

$objReader = new PHPEXcel_Reader_CSV();
$objPHPExcel = $objReader->load("sample.csv");

```

#### Setting CSV options

Often, CSV files are not really “comma separated”, or use semicolon (;) as a separator. You can instruct PHPEXcel\_Reader\_CSV some options before reading a CSV file.

Note that PHPEXcel\_Reader\_CSV by default assumes that the loaded CSV file is UTF-8 encoded. If you are reading CSV files that were created in Microsoft Office Excel the correct input encoding may rather be Windows-1252 (CP1252). Always make sure that the input encoding is set appropriately.

```

$objReader = new PHPEXcel_Reader_CSV();
$objReader->setInputEncoding('CP1252');
$objReader->setDelimiter(';');
$objReader->setEnclosure('');
$objReader->setLineEnding("\r\n");
$objReader->setSheetIndex(0);
$objPHPExcel = $objReader->load("sample.csv");

```

#### Read a specific worksheet

CSV files can only contain one worksheet. Therefore, you can specify which sheet to read from CSV:

```

$objReader->setSheetIndex(0);

```

#### Read into existing spreadsheet

When working with CSV files, it might occur that you want to import CSV data into an existing PHPEXcel object. The following code loads a CSV file into an existing \$objPHPExcel containing some sheets, and imports onto the 6<sup>th</sup> sheet:

```

$objReader = new PHPEXcel_Reader_CSV();
$objReader->setDelimiter(';');
$objReader->setEnclosure('');

```

```
$objReader->setLineEnding("\r\n");
$objReader->setSheetIndex(5);
$objReader->loadIntoExisting("05featuredemo.csv", $objPHPExcel);
```

## 6.7.2. PHPEXcel\_Writer\_CSV

### Writing a CSV file

You can write a .csv file using the following code:

```
$objWriter = new PHPEXcel_Writer_CSV($objPHPExcel);
$objWriter->save("05featuredemo.csv");
```

### Setting CSV options

Often, CSV files are not really “comma separated”, or use semicolon (;) as a separator. You can instruct PHPEXcel\_Writer\_CSV some options before writing a CSV file:

```
$objWriter = new PHPEXcel_Writer_CSV($objPHPExcel);
$objWriter->setDelimiter(';');
$objWriter->setEnclosure('');
$objWriter->setLineEnding("\r\n");
$objWriter->setSheetIndex(0);
$objWriter->save("05featuredemo.csv");
```

### Write a specific worksheet

CSV files can only contain one worksheet. Therefore, you can specify which sheet to write to CSV:

```
$objWriter->setSheetIndex(0);
```

### Formula pre-calculation

By default, this writer pre-calculates all formulas in the spreadsheet. This can be slow on large spreadsheets, and maybe even unwanted. You can however disable formula pre-calculation:

```
$objWriter = new PHPEXcel_Writer_CSV($objPHPExcel);
$objWriter->setPreCalculateFormulas(false);
$objWriter->save("05featuredemo.csv");
```

### Writing UTF-8 CSV files

A CSV file can be marked as UTF-8 by writing a BOM file header. This can be enabled by using the following code:

```
$objWriter = new PHPEXcel_Writer_CSV($objPHPExcel);
$objWriter->setUseBOM(true);
$objWriter->save("05featuredemo.csv");
```

### Decimal and thousands separators

If the worksheet you are exporting contains numbers with decimal or thousands separators then you should think about what characters you want to use for those before doing the export.

By default PHPEXcel looks up in the server’s locale settings to decide what characters to use. But to avoid problems it is recommended to set the characters explicitly as shown below.

English users will want to use this before doing the export:

```
require_once 'PHPExcel/Shared/String.php'
PHPExcel_Shared_String::setDecimalSeparator('.');
PHPExcel_Shared_String::setThousandsSeparator(',');
```

German users will want to use the opposite values.

```
require_once 'PHPExcel/Shared/String.php'
PHPExcel_Shared_String::setDecimalSeparator(',');
PHPExcel_Shared_String::setThousandsSeparator('.');
```

Note that the above code sets decimal and thousand separators as global options. This also affects how HTML and PDF is exported.

## 6.8. HTML

PHPExcel allows you to read or write a spreadsheet as HTML format, for quick representation of the data in it to anyone who does not have a spreadsheet application on their PC, or loading files saved by other scripts that simply create HTML markup and give it a .xls file extension.



### HTML limitations

Please note that HTML file format has some limits regarding to styling cells, number formatting, ...

### 6.8.1. PHPEXcel\_Reader\_HTML

#### Reading a spreadsheet

You can read an .html or .htm file using the following code:

```
$objReader = new PHPEXcel_Reader_HTML();
$objPHPExcel = $objReader->load("05featuredemo.html");
```



### HTML limitations

Please note that HTML reader is still experimental and does not yet support merged cells or nested tables cleanly

### 6.8.2. PHPEXcel\_Writer\_HTML



Please note that PHPEXcel\_Writer\_HTML only outputs the first worksheet by default.

#### Writing a spreadsheet

You can write a .htm file using the following code:

```
$objWriter = new PHPEXcel_Writer_HTML($objPHPExcel);
$objWriter->save("05featuredemo.htm");
```

#### Write all worksheets

HTML files can contain one or more worksheets. If you want to write all sheets into a single HTML file, use the following code:

```
$objWriter->writeAllSheets();
```

#### Write a specific worksheet

HTML files can contain one or more worksheets. Therefore, you can specify which sheet to write to HTML:

```
$objWriter->setSheetIndex(0);
```

#### Setting the images root of the HTML file

There might be situations where you want to explicitly set the included images root. For example, one might want to see `` instead of ``.

You can use the following code to achieve this result:

```
$objWriter->setImagesRoot('http://www.example.com');
```

#### Formula pre-calculation

By default, this writer pre-calculates all formulas in the spreadsheet. This can be slow on large spreadsheets, and maybe even unwanted. You can however disable formula pre-calculation:

```
$objWriter = new PHPEXcel_Writer_HTML($objPHPExcel);
$objWriter->setPreCalculateFormulas(false);
$objWriter->save("05featuredemo.htm");
```

## Embedding generated HTML in a web page

There might be a situation where you want to embed the generated HTML in an existing website. PHPEXcel\_Writer\_HTML provides support to generate only specific parts of the HTML code, which allows you to use these parts in your website.

Supported methods:

- generateHTMLHeader()
- generateStyles()
- generateSheetData()
- generateHTMLFooter()

Here's an example which retrieves all parts independently and merges them into a resulting HTML page:

```
<?php
$objWriter = new PHPEXcel_Writer_HTML($objPHPExcel);
echo $objWriter->generateHTMLHeader();
?>

<style>
<!--
html {
    font-family: Times New Roman;
    font-size: 9pt;
    background-color: white;
}

<?php
echo $objWriter->generateStyles(false); // do not write <style> and </style>
?>

-->
</style>

<?php
echo $objWriter->generateSheetData();
echo $objWriter->generateHTMLFooter();
?>
```

## Writing UTF-8 HTML files

A HTML file can be marked as UTF-8 by writing a BOM file header. This can be enabled by using the following code:

```
$objWriter = new PHPEXcel_Writer_HTML($objPHPExcel);
$objWriter->setUseBOM(true);
$objWriter->save("05featuredemo.htm");
```

## Decimal and thousands separators

See section PHPEXcel\_Writer\_CSV how to control the appearance of these.

## 6.9. PDF

PHPExcel allows you to write a spreadsheet into PDF format, for fast distribution of represented data.



### PDF limitations

Please note that PDF file format has some limits regarding to styling cells, number formatting, ...

### 6.9.1. PHPEXcel\_Writer\_PDF

PHPExcel's PDF Writer is a wrapper for a 3<sup>rd</sup>-Party PDF Rendering library such as tcPDF, mPDF or DomPDF. Prior to version 1.7.8 of PHPEXcel, the tcPDF library was bundled with PHPEXcel; but from

version 1.7.8 this was removed. Instead, you must now install a PDF Rendering library yourself; but PHPExcel will work with a number of different libraries. Currently, the following libraries are supported:

Library	Version used for testing	Downloadable from	PHPExcel Internal Constant
tcPDF	5.9	<a href="http://www.tcpdf.org/">http://www.tcpdf.org/</a>	PDF_RENDERER_TCPDF
mPDF	5.4	<a href="http://www.mpdf1.com/mpdf/">http://www.mpdf1.com/mpdf/</a>	PDF_RENDERER_MPDF
domPDF	0.6.0 beta 3	<a href="http://code.google.com/p/dompdf/">http://code.google.com/p/dompdf/</a>	PDF_RENDERER_DOMPDF

The different libraries have different strengths and weaknesses. Some generate better formatted output than others, some are faster or use less memory than others, while some generate smaller .pdf files. It is the developers choice which one they wish to use, appropriate to their own circumstances.

Before instantiating a Writer to generate PDF output, you will need to indicate which Rendering library you are using, and where it is located.

```
$rendererName = PHPExcel_Settings::PDF_RENDERER_MPDF;
$rendererLibrary = 'mPDF5.4';
$rendererLibraryPath = dirname(__FILE__) . '/../..../libraries/PDF/' .
$rendererLibrary;
if (!PHPExcel_Settings::setPdfRenderer(
    $rendererName,
    $rendererLibraryPath
)) {
    die(
        'Please set the $rendererName and $rendererLibraryPath values' .
        PHP_EOL .
        ' as appropriate for your directory structure'
    );
}
```

## Writing a spreadsheet

Once you have identified the Renderer that you wish to use for PDF generation, you can write a .pdf file using the following code:

```
$objWriter = new PHPExcel_Writer_PDF($objPHPExcel);
$objWriter->save("05featuredemo.pdf");
```



Please note that PHPExcel\_Writer\_PDF only outputs the first worksheet by default.

## Write all worksheets

PDF files can contain one or more worksheets. If you want to write all sheets into a single PDF file, use the following code:

```
$objWriter->writeAllSheets();
```

## Write a specific worksheet

PDF files can contain one or more worksheets. Therefore, you can specify which sheet to write to PDF:

```
$objWriter->setSheetIndex(0);
```

## Formula pre-calculation

By default, this writer pre-calculates all formulas in the spreadsheet. This can be slow on large spreadsheets, and maybe even unwanted. You can however disable formula pre-calculation:

```
$objWriter = new PHPExcel_Writer_PDF($objPHPExcel);
$objWriter->setPreCalculateFormulas(false);
$objWriter->save("05featuredemo.pdf");
```

### Decimal and thousands separators

See section `PHPExcel_Writer_CSV` how to control the appearance of these.

## 6.10. *Generating Excel files from templates (read, modify, write)*

Readers and writers are the tools that allow you to generate Excel files from templates. This requires less coding effort than generating the Excel file from scratch, especially if your template has many styles, page setup properties, headers etc.

Here is an example how to open a template file, fill in a couple of fields and save it again:

```
$objPHPExcel = PHPExcel_IOFactory::load('template.xlsx');

$objWorksheet = $objPHPExcel->getActiveSheet();
$objWorksheet->getCell('A1')->setValue('John');
$objWorksheet->getCell('A2')->setValue('Smith');

$objWriter = PHPExcel_IOFactory::createWriter($objPHPExcel, 'Excel5');
$objWriter->save('write.xls');
```

Notice that it is ok to load an `xlsx` file and generate an `xls` file.

## 7. Credits

Please refer to the internet page

<http://www.codeplex.com/PHPExcel/Wiki/View.aspx?title=Credits&referringTitle=Home> for up-to-date credits.

## Appendix A: Valid array keys for style applyFromArray()

The following table lists the valid array keys for PHPExcel\_Style applyFromArray() classes. If the “Maps to property” column maps a key to a setter, the value provided for that key will be applied directly. If the “Maps to property” column maps a key to a getter, the value provided for that key will be applied as another style array.

PHPExcel_Style	
Array key:	Maps to property:
fill	getFill()
font	getFont()
borders	getBorders()
alignment	getAlignment()
numberformat	getNumberFormat()
protection	getProtection()
PHPExcel_Style_Fill	
Array key:	Maps to property:
type	setFillType()
rotation	setRotation()
startcolor	getStartColor()
endcolor	getEndColor()
color	getStartColor()
PHPExcel_Style_Font	
Array key:	Maps to property:
name	setName()
bold	setBold()
italic	setItalic()
underline	setUnderline()
strike	setStrikethrough()
color	getColor()
size	setSize()
superScript	setSuperScript()
subScript	setSubScript()
PHPExcel_Style_Borders	
Array key:	Maps to property:
allborders	getLeft(); getRight(); getTop(); getBottom()
left	getLeft()
right	getRight()
top	getTop()
bottom	getBottom()
diagonal	getDiagonal()
vertical	getVertical()
horizontal	getHorizontal()
diagonaldirection	setDiagonalDirection()
outline	setOutline()
PHPExcel_Style_Border	
Array key:	Maps to property:
style	setBorderStyle()
color	getColor()
PHPExcel_Style_Alignment	
Array key:	Maps to property:
horizontal	setHorizontal()
vertical	setVertical()

rotation wrap shrinkToFit indent	setTextRotation() setWrapText() setShrinkToFit() setIndent()
<b>PHPExcel_Style_NumberFormat</b>	
Array key: code	Maps to property: setFormatCode()
<b>PHPExcel_Style_Protection</b>	
Array key: locked hidden	Maps to property: setLocked() setHidden()