

# **PHPExcel User Documentation – Reading Spreadsheet Files**

**Author:** Mark Baker  
**Version:** 1.7.4  
**Date:** 11 October 2012

## Contents

|   |    |
|---|----|
| PHPExcel User Documentation - Reading Spreadsheet Files .....   | 1  |
| 1. Spreadsheet File Formats .....   | 1  |
| 2. Loading a Spreadsheet File .....   | 3  |
| 3. Creating a Reader and Loading a Spreadsheet File .....   | 4  |
| 4. Spreadsheet Reader Options .....   | 5  |
| 4.1. Reading Only Data from a Spreadsheet File .....  | 5  |
| 4.2. Reading Only Named Worksheets from a File .....  | 6  |
| 4.3. Reading Only Specific Columns and Rows from a File (Read Filters).....   | 7  |
| 4.4. Combining Multiple Files into a Single PHPExcel Object .....   | 10 |
| 4.5. Combining Read Filters with the setSheetIndex() method to split a large CSV file across multiple Worksheets..... | 11 |
| 4.6. Pipe or Tab Separated Value Files .....  | 13 |
| 4.7. A Brief Word about the Advanced Value Binder.....  | 14 |
| 5. Error Handling.....  | 15 |

# 1. Spreadsheet File Formats

PHPExcel can read a number of different spreadsheet file formats, although not all features are supported by all of the readers. Check the Functionality Cross-Reference document (Functionality Cross-Reference.xls) for a list that identifies which features are supported by which readers.

Currently, PHPExcel supports the following File Types for Reading:

## Excel5

The Microsoft Excel™ Binary file format (BIFF5 and BIFF8) is a binary file format that was used by Microsoft Excel™ between versions 95 and 2003. The format is supported (to various extents) by most spreadsheet programs. BIFF files normally have an extension of .xls. Documentation describing the format can be found online at [http://msdn.microsoft.com/en-us/library/cc313154\(v=office.12\).aspx](http://msdn.microsoft.com/en-us/library/cc313154(v=office.12).aspx) or from [http://download.microsoft.com/download/2/4/8/24862317-78F0-4C4B-B355-C7B2C1D997DB/\[MS-XLS\].pdf](http://download.microsoft.com/download/2/4/8/24862317-78F0-4C4B-B355-C7B2C1D997DB/[MS-XLS].pdf) (as a downloadable PDF).

## Excel2003XML

Microsoft Excel™ 2003 included options for a file format called SpreadsheetML. This file is a zipped XML document. It is not very common, but its core features are supported.

## Excel2007

Microsoft Excel™ 2007 shipped with a new file format, namely Microsoft Office Open XML SpreadsheetML, and Excel 2010 extended this still further with its new features such as sparklines. These files typically have an extension of .xlsx. This format is based around a zipped collection of eXtensible Markup Language (XML) files. Microsoft Office Open XML SpreadsheetML is mostly standardized in ECMA 376 ([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)) and ISO 29500.

## OOCalc

aka Open Document Format (ODF) or OASIS, this is the OpenOffice.org XML File Format for spreadsheets. It comprises a zip archive including several components all of which are text files, most of these with markup in the eXtensible Markup Language (XML). It is the standard file format for OpenOffice.org Calc and StarCalc, and files typically have an extension of .ods. The published specification for the file format is available from the OASIS Open Office XML Format Technical Committee web page ([http://www.oasis-open.org/committees/tc\\_home.php?wg\\_abbrev=office#technical](http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=office#technical)). Other information is available from the OpenOffice.org XML File Format web page (<http://xml.openoffice.org/general.html>), part of the OpenOffice.org project.

## SYLK

This is the Microsoft Multiplan Symbolic Link Interchange (SYLK) file format. Multiplan was a predecessor to Microsoft Excel™. Files normally have an extension of .slk. While not common, there are still a few applications that generate SYLK files as a cross-platform option, because (despite being limited to a single worksheet) it is a simple format to implement, and supports some basic data and cell formatting options (unlike CSV files).

## Gnumeric

The Gnumeric file format is used by the Gnome Gnumeric spreadsheet application, and typically files have an extension of .gnumeric. The file contents are stored using eXtensible Markup Language (XML) markup, and the file is then compressed using the GNU project's gzip compression library. <http://projects.gnome.org/gnumeric/doc/file-format-gnumeric.shtml>

## CSV

Comma Separated Value (CSV) file format is a common structuring strategy for text format files. In CSV files, each line in the file represents a row of data and (within each line of the file) the different data fields (or columns) are separated from one another using a

comma (","). If a data field contains a comma, then it should be enclosed (typically in quotation marks ("")). Sometimes tabs "\t" or the pipe symbol ("|") are used as separators instead of a comma. Because CSV is a text-only format, it doesn't support any data formatting options.

## 2. Loading a Spreadsheet File

The simplest way to load a workbook file is to let PHPExcel's IO Factory identify the file type and load it, calling the static load() method of the PHPExcel\_IOFactory class.

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

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

See [Examples/Reader/exampleReader01.php](#) for a working example of this code.

The load() method will attempt to identify the file type, and instantiate a loader for that file type; using it to load the file and store the data and any formatting in a PHPExcel object.

The method makes an initial guess at the loader to instantiate based on the file extension; but will test the file before actually executing the load: so if (for example) the file is actually a CSV file that has been given a .xls extension (quite a common practise), it will reject the Excel5 loader that it would normally use for a .xls file; and test the file using the other loaders until it finds the appropriate loader, and then use that to read the file.

While easy to implement in your code, and you don't need to worry about the file type; this isn't the most efficient method to load a file; and it lacks the flexibility to configure the loader in any way before actually reading the file into a PHPExcel object.

### 3. Creating a Reader and Loading a Spreadsheet File

If you know the file type of the spreadsheet file that you need to load, you can instantiate a new reader object for that file type, then use the reader's load() method to read the file to a PHPEXcel object. It is possible to instantiate the reader objects for each of the different supported filetype by name. However, you may get unpredictable results if the file isn't of the right type (e.g. it is a CSV with an extension of .xls), although this type of exception should normally be trapped.

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

/** Create a new Excel5 Reader */
$objReader = new PHPEXcel_Reader_Excel5();
// $objReader = new PHPEXcel_Reader_Excel2007();
// $objReader = new PHPEXcel_Reader_Excel2003XML();
// $objReader = new PHPEXcel_Reader_OOCalc();
// $objReader = new PHPEXcel_Reader_SYLK();
// $objReader = new PHPEXcel_Reader_Gnumeric();
// $objReader = new PHPEXcel_Reader_CSV();
/** Load $inputFileName to a PHPEXcel Object */
$objPHPExcel = $objReader->load($inputFileName);
```

See Examples/Reader/exampleReader02.php for a working example of this code.

Alternatively, you can use the IO Factory's createReader() method to instantiate the reader object for you, simply telling it the file type of the reader that you want instantiating.

```
$inputFileType = 'Excel5';
// $inputFileType = 'Excel2007';
// $inputFileType = 'Excel2003XML';
// $inputFileType = 'OOCalc';
// $inputFileType = 'SYLK';
// $inputFileType = 'Gnumeric';
// $inputFileType = 'CSV';
$inputFileName = './sampleData/example1.xls';

/** Create a new Reader of the type defined in $inputFileType */
$objReader = PHPEXcel_IOFactory::createReader($inputFileType);
/** Load $inputFileName to a PHPEXcel Object */
$objPHPExcel = $objReader->load($inputFileName);
```

See Examples/Reader/exampleReader03.php for a working example of this code.

If you're uncertain of the filetype, you can use the IO Factory's identify() method to identify the reader that you need, before using the createReader() method to instantiate the reader object.

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

/** Identify the type of $inputFileName */
$inputFileType = PHPEXcel_IOFactory::identify($inputFileName);
/** Create a new Reader of the type that has been identified */
$objReader = PHPEXcel_IOFactory::createReader($inputFileType);
/** Load $inputFileName to a PHPEXcel Object */
$objPHPExcel = $objReader->load($inputFileName);
```

See Examples/Reader/exampleReader04.php for a working example of this code.

## 4. Spreadsheet Reader Options

Once you have created a reader object for the workbook that you want to load, you have the opportunity to set additional options before executing the load() method.

### 4.1. *Reading Only Data from a Spreadsheet File*

If you're only interested in the cell values in a workbook, but don't need any of the cell formatting information, then you can set the reader to read only the data values and any formulae from each cell using the setReadDataOnly() method.

```
$inputFileType = 'Excel5';
$inputFileName = './sampleData/example1.xls';

/** Create a new Reader of the type defined in $inputFileType */
$objReader = PHPExcel_IOFactory::createReader($inputFileType);
/** Advise the Reader that we only want to load cell data */
$objReader->setReadDataOnly(true);
/** Load $inputFileName to a PHPExcel Object */
$objPHPExcel = $objReader->load($inputFileName);
```

See [Examples/Reader/exampleReader05.php](#) for a working example of this code.

It is important to note that Workbooks (and PHPExcel) store dates and times as simple numeric values: they can only be distinguished from other numeric values by the format mask that is applied to that cell. When setting read data only to true, PHPExcel doesn't read the cell format masks, so it is not possible to differentiate between dates/times and numbers.

The Gnumeric loader has been written to read the format masks for date values even when read data only has been set to true, so it can differentiate between dates/times and numbers; but this change hasn't yet been implemented for the other readers.

Reading Only Data from a Spreadsheet File applies to Readers:

|           |     |        |     |              |     |
|-----------|-----|--------|-----|--------------|-----|
| Excel2007 | YES | Excel5 | YES | Excel2003XML | YES |
| OOCalc    | YES | SYLK   | NO  | Gnumeric     | YES |
| CSV       | NO  |        |     |              |     |

## 4.2. Reading Only Named WorkSheets from a File

If your workbook contains a number of worksheets, but you are only interested in reading some of those, then you can use the `setLoadSheetsOnly()` method to identify those sheets you are interested in reading.

To read a single sheet, you can pass that sheet name as a parameter to the `setLoadSheetsOnly()` method.

```
$inputFileType = 'Excel5';
$inputFileName = './sampleData/example1.xls';
$sheetname = 'Data Sheet #2';

/** Create a new Reader of the type defined in $inputFileType */
$objReader = PHPExcel_IOFactory::createReader($inputFileType);
/** Advise the Reader of which WorkSheets we want to load */
$objReader->setLoadSheetsOnly($sheetname);
/** Load $inputFileName to a PHPExcel Object */
$objPHPExcel = $objReader->load($inputFileName);
```

See [Examples/Reader/exampleReader07.php](#) for a working example of this code.

If you want to read more than just a single sheet, you can pass a list of sheet names as an array parameter to the `setLoadSheetsOnly()` method.

```
$inputFileType = 'Excel5';
$inputFileName = './sampleData/example1.xls';
$sheetnames = array('Data Sheet #1', 'Data Sheet #3');

/** Create a new Reader of the type defined in $inputFileType */
$objReader = PHPExcel_IOFactory::createReader($inputFileType);
/** Advise the Reader of which WorkSheets we want to load */
$objReader->setLoadSheetsOnly($sheetnames);
/** Load $inputFileName to a PHPExcel Object */
$objPHPExcel = $objReader->load($inputFileName);
```

See [Examples/Reader/exampleReader08.php](#) for a working example of this code.

To reset this option to the default, you can call the `setLoadAllSheets()` method.

```
$inputFileType = 'Excel5';
$inputFileName = './sampleData/example1.xls';

/** Create a new Reader of the type defined in $inputFileType */
$objReader = PHPExcel_IOFactory::createReader($inputFileType);
/** Advise the Reader to load all Worksheets */
$objReader->setLoadAllSheets();
/** Load $inputFileName to a PHPExcel Object */
$objPHPExcel = $objReader->load($inputFileName);
```

See [Examples/Reader/exampleReader06.php](#) for a working example of this code.

Reading Only Named WorkSheets from a File applies to Readers:

|           |     |        |     |              |     |
|-----------|-----|--------|-----|--------------|-----|
| Excel2007 | YES | Excel5 | YES | Excel2003XML | YES |
| OOCalc    | YES | SYLK   | NO  | Gnumeric     | YES |
| CSV       | NO  |        |     |              |     |



### 4.3. Reading Only Specific Columns and Rows from a File (Read Filters)

If you are only interested in reading part of a worksheet, then you can write a filter class that identifies whether or not individual cells should be read by the loader. A read filter must implement the `PHPExcel_Reader_IReadFilter` interface, and contain a `readCell()` method that accepts arguments of `$column`, `$row` and `$worksheetName`, and return a boolean true or false that indicates whether a workbook cell identified by those arguments should be read or not.

```
$inputFileType = 'Excel5';
$inputFileName = './sampleData/example1.xls';
$sheetname = 'Data Sheet #3';

/** Define a Read Filter class implementing PHPExcel_Reader_IReadFilter */
/
class MyReadFilter implements PHPExcel_Reader_IReadFilter
{
    public function readCell($column, $row, $worksheetName = '') {
        // Read rows 1 to 7 and columns A to E only
        if ($row >= 1 && $row <= 7) {
            if (in_array($column,range('A','E'))){
                return true;
            }
        }
        return false;
    }
}

/** Create an Instance of our Read Filter */
$filterSubset = new MyReadFilter();

/** Create a new Reader of the type defined in $inputFileType */
$objReader = PHPExcel_IOFactory::createReader($inputFileType);
/** Tell the Reader that we want to use the Read Filter */
$objReader->setReadFilter($filterSubset);
/** Load only the rows and columns that match our filter to PHPExcel */
$objPHPExcel = $objReader->load($inputFileName);
```

See [Examples/Reader/exampleReader09.php](#) for a working example of this code.

This example is not particularly useful, because it can only be used in a very specific circumstance (when you only want cells in the range A1:E7 from your worksheet. A generic Read Filter would probably be more useful:

```
/** Define a Read Filter class implementing PHPExcel_Reader_IReadFilter */
/
class MyReadFilter implements PHPExcel_Reader_IReadFilter
{
    private $_startRow = 0;
    private $_endRow = 0;
    private $_columns = array();

    /** Get the list of rows and columns to read */
    public function __construct($startRow, $endRow, $columns) {
        $this->_startRow = $startRow;
        $this->_endRow = $endRow;
        $this->_columns = $columns;
    }
}
```

```

    public function readCell($column, $row, $worksheetName = '') {
        //  Only read the rows and columns that were configured
        if ($row >= $this->_startRow && $row <= $this->_endRow) {
            if (in_array($column,$this->_columns)) {
                return true;
            }
        }
        return false;
    }
}

/** Create an Instance of our Read Filter, passing in the cell range */
$filterSubset = new MyReadFilter(9,15,range('G','K'));

```

See [Examples/Reader/exampleReader10.php](#) for a working example of this code.

This can be particularly useful for conserving memory, by allowing you to read and process a large workbook in “chunks”: an example of this usage might be when transferring data from an Excel worksheet to a database.

```

$inputFileType = 'Excel5';
$inputFileName = './sampleData/example2.xls';

/** Define a Read Filter class implementing PHPExcel_Reader_IReadFilter */
class chunkReadFilter implements PHPExcel_Reader_IReadFilter
{
    private $_startRow = 0;
    private $_endRow   = 0;

    /** Set the list of rows that we want to read */
    public function setRows($startRow, $chunkSize) {
        $this->_startRow = $startRow;
        $this->_endRow   = $startRow + $chunkSize;
    }

    public function readCell($column, $row, $worksheetName = '') {
        //  Only read the heading row, and the configured rows
        if (($row == 1) ||
            ($row >= $this->_startRow && $row < $this->_endRow)) {
            return true;
        }
        return false;
    }
}

/** Create a new Reader of the type defined in $inputFileType */
$objReader = PHPExcel_IOFactory::createReader($inputFileType);

/** Define how many rows we want to read for each "chunk" */
$chunkSize = 2048;
/** Create a new Instance of our Read Filter */
$chunkFilter = new chunkReadFilter();

/** Tell the Reader that we want to use the Read Filter */
$objReader->setReadFilter($chunkFilter);

```

```
/** Loop to read our worksheet in "chunk size" blocks */  
for ($startRow = 2; $startRow <= 65536; $startRow += $chunkSize) {  
    /** Tell the Read Filter which rows we want this iteration */  
    $chunkFilter->setRows($startRow,$chunkSize);  
    /** Load only the rows that match our filter */  
    $objPHPExcel = $objReader->load($inputFileName);  
    // Do some processing here  
}
```

See [Examples/Reader/exampleReader12.php](#) for a working example of this code.

Using Read Filters applies to:

|           |     |        |     |              |     |
|-----------|-----|--------|-----|--------------|-----|
| Excel2007 | YES | Excel5 | YES | Excel2003XML | YES |
| OOCalc    | YES | SYLK   | NO  | Gnumeric     | YES |
| CSV       | YES |        |     |              |     |

#### 4.4. Combining Multiple Files into a Single PHPEXcel Object

While you can limit the number of worksheets that are read from a workbook file using the `setLoadSheetsOnly()` method, certain readers also allow you to combine several individual “sheets” from different files into a single PHPEXcel object, where each individual file is a single worksheet within that workbook. For each file that you read, you need to indicate which worksheet index it should be loaded into using the `setSheetIndex()` method of the `$objReader`, then use the `loadIntoExisting()` method rather than the `load()` method to actually read the file into that worksheet.

```
$inputFileType = 'CSV';
$inputFileNames = array('./sampleData/example1.csv',
                        './sampleData/example2.csv',
                        './sampleData/example3.csv'
                        );

/** Create a new Reader of the type defined in $inputFileType */
$objReader = PHPEXcel_IOFactory::createReader($inputFileType);

/** Extract the first named file from the array list */
$inputFileName = array_shift($inputFileNames);
/** Load the initial file to the first worksheet in a PHPEXcel Object */
$objPHPExcel = $objReader->load($inputFileName);
/** Set the worksheet title (to the filename that we've loaded) */
$objPHPExcel->getActiveSheet()
    ->setTitle(pathinfo($inputFileName, PATHINFO_BASENAME));

/** Loop through all the remaining files in the list */
foreach($inputFileNames as $sheet => $inputFileName) {
    /** Increment the worksheet index pointer for the Reader */
    $objReader->setSheetIndex($sheet+1);
    /** Load the current file into a new worksheet in PHPEXcel */
    $objReader->loadIntoExisting($inputFileName, $objPHPExcel);
    /** Set the worksheet title (to the filename that we've loaded) */
    $objPHPExcel->getActiveSheet()
        ->setTitle(pathinfo($inputFileName, PATHINFO_BASENAME));
}
```

See [Examples/Reader/exampleReader13.php](#) for a working example of this code.

Note that using the same sheet index for multiple sheets won't append files into the same sheet, but overwrite the results of the previous load. You cannot load multiple CSV files into the same worksheet.

Combining Multiple Files into a Single PHPEXcel Object applies to:

|           |     |        |     |              |    |
|-----------|-----|--------|-----|--------------|----|
| Excel2007 | NO  | Excel5 | NO  | Excel2003XML | NO |
| OOCalc    | NO  | SYLK   | YES | Gnumeric     | NO |
| CSV       | YES |        |     |              |    |

#### 4.5. Combining Read Filters with the setSheetIndex() method to split a large CSV file across multiple Worksheets

An Excel5 BIFF .xls file is limited to 65536 rows in a worksheet, while the Excel2007 Microsoft Office Open XML SpreadsheetML .xlsx file is limited to 1,048,576 rows in a worksheet; but a CSV file is not limited other than by available disk space. This means that we wouldn't ordinarily be able to read all the rows from a very large CSV file that exceeded those limits, and save it as an Excel5 or Excel2007 file. However, by using Read Filters to read the CSV file in "chunks" (using the chunkReadFilter Class that we defined in section 4.3 above), and the setSheetIndex() method of the \$objReader, we can split the CSV file across several individual worksheets.

```
$inputFileType = 'CSV';
$inputFileName = './sampleData/example2.csv';

echo 'Loading file ',pathinfo($inputFileName,PATHINFO_BASENAME),' using IOF
actory with a defined reader type of ', $inputFileType, '<br />';
/** Create a new Reader of the type defined in $inputFileType */
$objReader = PHPExcel_IOFactory::createReader($inputFileType);

/** Define how many rows we want to read for each "chunk" */
$chunkSize = 65530;
/** Create a new Instance of our Read Filter */
$chunkFilter = new chunkReadFilter();

/** Tell the Reader that we want to use the Read Filter */
/** and that we want to store it in contiguous rows/columns */
$objReader->setReadFilter($chunkFilter)
->setContiguous(true);

/** Instantiate a new PHPExcel object manually */
$objPHPExcel = new PHPExcel();

/** Set a sheet index */
$sheet = 0;
/** Loop to read our worksheet in "chunk size" blocks */
/** $startRow is set to 2 initially because we always read the headings
in row #1 */
for ($startRow = 2; $startRow <= 1000000; $startRow += $chunkSize) {
    /** Tell the Read Filter which rows we want to read this loop */
    $chunkFilter->setRows($startRow,$chunkSize);

    /** Increment the worksheet index pointer for the Reader */
    $objReader->setSheetIndex($sheet);
    /** Load only the rows that match our filter into a new worksheet */
    $objReader->loadIntoExisting($inputFileName,$objPHPExcel);
    /** Set the worksheet title for the sheet that we've justloaded) */
    /** and increment the sheet index as well */
    $objPHPExcel->getActiveSheet()->setTitle('Country Data #'.(++$sheet));
}
```

See Examples/Reader/exampleReader14.php for a working example of this code.

This code will read 65,530 rows at a time from the CSV file that we're loading, and store each "chunk" in a new worksheet.

The setContiguous() method for the Reader is important here. It is applicable only when working with a Read Filter, and identifies whether or not the cells should be stored by their position within the CSV file, or their position relative to the filter.

For example, if the filter returned true for cells in the range B2:C3, then with setContiguous set to false (the default) these would be loaded as B2:C3 in the PHPExcel object; but with setContiguous set to true, they would be loaded as A1:B2.

Splitting a single loaded file across multiple worksheets applies to:

|           |     |        |    |              |    |
|-----------|-----|--------|----|--------------|----|
| Excel2007 | NO  | Excel5 | NO | Excel2003XML | NO |
| OOCalc    | NO  | SYLK   | NO | Gnumeric     | NO |
| CSV       | YES |        |    |              |    |

## 4.6. Pipe or Tab Separated Value Files

The CSV loader defaults to loading a file where comma is used as the separator, but you can modify this to load tab- or pipe-separated value files using the `setDelimiter()` method.

```
$inputFileType = 'CSV';
$inputFileName = './sampleData/example1.tsv';

/** Create a new Reader of the type defined in $inputFileType */
$objReader = PHPExcel_IOFactory::createReader($inputFileType);
/** Set the delimiter to a TAB character */
$objReader->setDelimiter("\t");
//    $objReader->setDelimiter('|');

/** Load the file to a PHPExcel Object */
$objPHPExcel = $objReader->load($inputFileName);
```

See [Examples/Reader/exampleReader15.php](#) for a working example of this code.

In addition to the delimiter, you can also use the following methods to set other attributes for the data load:

|                                 |                    |
|---------------------------------|--------------------|
| <code>setEnclosure()</code>     | default is "       |
| <code>setLineEnding()</code>    | default is PHP_EOL |
| <code>setInputEncoding()</code> | default is UTF-8   |

Setting CSV delimiter applies to:

|           |     |        |    |              |    |
|-----------|-----|--------|----|--------------|----|
| Excel2007 | NO  | Excel5 | NO | Excel2003XML | NO |
| OOCalc    | NO  | SYLK   | NO | Gnumeric     | NO |
| CSV       | YES |        |    |              |    |

#### 4.7. A Brief Word about the Advanced Value Binder

When loading data from a file that contains no formatting information, such as a CSV file, then data is read either as strings or numbers (float or integer). This means that PHPEXcel does not automatically recognise dates/times (such as "16-Apr-2009" or "13:30"), booleans ("TRUE" or "FALSE"), percentages ("75%"), hyperlinks ("http://www.phpexcel.net"), etc as anything other than simple strings. However, you can apply additional processing that is executed against these values during the load process within a Value Binder.

A Value Binder is a class that implement the PHPEXcel\_Cell\_IValueBinder interface. It must contain a bindValue() method that accepts a PHPEXcel\_Cell and a value as arguments, and return a boolean true or false that indicates whether the workbook cell has been populated with the value or not. The Advanced Value Binder implements such a class: amongst other tests, it identifies a string comprising "TRUE" or "FALSE" (based on locale settings) and sets it to a boolean; or a number in scientific format (e.g. "1.234e-5") and converts it to a float; or dates and times, converting them to their Excel timestamp value - before storing the value in the cell object. It also sets formatting for strings that are identified as dates, times or percentages. It could easily be extended to provide additional handling (including text or cell formatting) when it encountered a hyperlink, or HTML markup within a CSV file.

So using a Value Binder allows a great deal more flexibility in the loader logic when reading unformatted text files.

```
/** Tell PHPEXcel that we want to use the Advanced Value Binder */
PHPExcel_Cell::setValueBinder( new PHPEXcel_Cell_AdvancedValueBinder() );

$inputFileType = 'CSV';
$inputFileName = './sampleData/example1.tsv';

$objReader = PHPEXcel_IOFactory::createReader($inputFileType);
$objReader->setDelimiter("\t");
$objPHPExcel = $objReader->load($inputFileName);
```

See Examples/Reader/exampleReader15.php for a working example of this code.

Loading using a Value Binder applies to:

|           |     |        |    |              |    |
|-----------|-----|--------|----|--------------|----|
| Excel2007 | NO  | Excel5 | NO | Excel2003XML | NO |
| OOCalc    | NO  | SYLK   | NO | Gnumeric     | NO |
| CSV       | YES |        |    |              |    |



## 5. Error Handling

Of course, you should always apply some error handling to your scripts as well. PHPExcel throws exceptions, so you can wrap all your code that accesses the library methods within Try/Catch blocks to trap for any problems that are encountered, and deal with them in an appropriate manner.

```
$inputFileName = './sampleData/example-1.xls';

try {
    /** Load $inputFileName to a PHPExcel Object */
    $objPHPExcel = PHPExcel_IOFactory::load($inputFileName);
} catch(Exception $e) {
    die('Error loading file: '.$e->getMessage());
}
```

See [Examples/Reader/exampleReader16.php](#) for a working example of this code.