**ExcelDataReader**

Lightweight and fast library written in C# for reading Microsoft Excel files (2.0-2007).

Please feel free to fork and submit pull requests to the develop branch.

If you are reporting an issue it is really useful if you can supply an example Excel file as this makes debugging much easier and without it we may not be able to resolve any problems.

**Supported file formats and versions**

| **File Type** | **Container Format** | **File Format** | **Excel Version(s)** |
| --- | --- | --- | --- |
| .xlsx | ZIP, CFB+ZIP | OpenXml | 2007 and newer |
| .xls | CFB | BIFF8 | 97, 2000, XP, 2003 98, 2001, v.X, 2004 (Mac) |
| .xls | CFB | BIFF5 | 5.0, 95 |
| .xls | - | BIFF4 | 4.0 |
| .xls | - | BIFF3 | 3.0 |
| .xls | - | BIFF2 | 2.0, 2.2 |
| .csv | - | CSV | (All) |

**Finding the binaries**

It is recommended to use NuGet. F.ex through the VS Package Manager Console Install-Package <package> or using the VS "Manage NuGet Packages..." extension.

As of ExcelDataReader version 3.0, the project was split into multiple packages:

Install the ExcelDataReader base package to use the "low level" reader interface. Compatible with net20, net45, netstandard1.3 and netstandard2.0.

Install the ExcelDataReader.DataSet extension package to use the AsDataSet() method to populate a System.Data.DataSet. This will also pull in the base package. Compatible with net20, net45 and netstandard2.0.

**How to use**

using (var stream = File.Open(filePath, FileMode.Open, FileAccess.Read)) {

// Auto-detect format, supports:

// - Binary Excel files (2.0-2003 format; \*.xls)

// - OpenXml Excel files (2007 format; \*.xlsx)

using (var reader = ExcelReaderFactory.CreateReader(stream)) {

// Choose one of either 1 or 2:

// 1. Use the reader methods

do {

while (reader.Read()) {

// reader.GetDouble(0);

}

} while (reader.NextResult());

// 2. Use the AsDataSet extension method

var result = reader.AsDataSet();

// The result of each spreadsheet is in result.Tables

}

}

**Reading .CSV files**

Use ExcelReaderFactory.CreateCsvReader instead of CreateReader to parse a stream of plain text with comma separated values.

See also the configuration options FallbackEncoding and AutodetectSeparators.

The input CSV is always parsed once completely to set FieldCount, RowCount, Encoding, Separator (or twice if the CSV lacks BOM and is not UTF8), and then parsed once again while iterating the row records. Throws System.Text.DecoderFallbackException if the input cannot be parsed with the specified encoding.

The reader returns all CSV field values as strings and makes no attempts to convert the data to numbers or dates. This caller is responsible for interpreting the CSV data.

**Using the reader methods**

The AsDataSet() extension method is a convenient helper for quickly getting the data, but is not always available or desirable to use. IExcelDataReader extends the System.Data.IDataReader and IDataRecord interfaces to navigate and retrieve data at a lower level. The most important reader methods and properties:

* Read() reads a row from the current sheet.
* NextResult() advances the cursor to the next sheet.
* ResultsCount returns the number of sheets in the current workbook.
* Name returns the name of the current sheet.
* CodeName returns the VBA code name identifier of the current sheet.
* FieldCount returns the number of columns in the current sheet.
* RowCount returns the number of rows in the current sheet. This includes terminal empty rows which are otherwise excluded by AsDataSet().
* HeaderFooter returns an object with information about the headers and footers, or null if there are none.
* MergeCells returns an array of merged cell ranges in the current sheet.
* RowHeight returns the visual height of the current row in points. May be 0 if the row is hidden.
* GetFieldType() returns the type of a value in the current row. Always one of the types supported by Excel: double, int, bool, DateTime, TimeSpan, string, or null if there is no value.
* IsDBNull() checks if a value in the current row is null.
* GetValue() returns a value from the current row as an object, or null if there is no value.
* GetDouble(), GetInt32(), GetBoolean(), GetDateTime(), GetString() return a value from the current row cast to their respective type.
* GetNumberFormatString() returns a string containing the formatting codes for a value in the current row, or null if there is no value. See also the Formatting section below.
* The typed Get\*() methods throw InvalidCastException unless the types match exactly.

**CreateReader() configuration options**

The ExcelReaderFactory.CreateReader(), CreateBinaryReader(), CreateOpenXmlReader(), CreateCsvReader() methods accept an optional configuration object to modify the behavior of the reader:

var reader = ExcelReaderFactory.CreateReader(stream, new ExcelReaderConfiguration() {

// Gets or sets the encoding to use when the input XLS lacks a CodePage

// record, or when the input CSV lacks a BOM and does not parse as UTF8.

// Default: cp1252. (XLS BIFF2-5 and CSV only)

FallbackEncoding = Encoding.GetEncoding(1252),

// Gets or sets the password used to open password protected workbooks.

Password = "password",

// Gets or sets an array of CSV separator candidates. The reader

// autodetects which best fits the input data. Default: , ; TAB | #

// (CSV only)

AutodetectSeparators = new char[] { ',', ';', '\t', '|', '#' };

});

**AsDataSet() configuration options**

The AsDataSet() method accepts an optional configuration object to modify the behavior of the DataSet conversion:

var result = reader.AsDataSet(new ExcelDataSetConfiguration() {

// Gets or sets a value indicating whether to set the DataColumn.DataType

// property in a second pass.

UseColumnDataType = true,

// Gets or sets a callback to obtain configuration options for a DataTable.

ConfigureDataTable = (tableReader) => new ExcelDataTableConfiguration() {

// Gets or sets a value indicating the prefix of generated column names.

EmptyColumnNamePrefix = "Column",

// Gets or sets a value indicating whether to use a row from the

// data as column names.

UseHeaderRow = false,

// Gets or sets a callback to determine which row is the header row.

// Only called when UseHeaderRow = true.

ReadHeaderRow = (rowReader) => {

// F.ex skip the first row and use the 2nd row as column headers:

rowReader.Read();

},

// Gets or sets a callback to determine whether to include the

// current row in the DataTable.

FilterRow = (rowReader) => {

return true;

},

// Gets or sets a callback to determine whether to include the specific

// column in the DataTable. Called once per column after reading the

// headers.

FilterColumn = (rowReader, columnIndex) => {

return true;

}

}

});

**Formatting**

ExcelDataReader does not support formatting directly. Users may retreive the number format string for a cell through IExcelDataReader.GetNumberFormatString(i) and use the third party ExcelNumberFormat library for formatting purposes.

Example helper method using ExcelDataReader and ExcelNumberFormat to format a value:

string GetFormattedValue(IExcelDataReader reader, int columnIndex, CultureInfo culture) {

var value = reader.GetValue(columnIndex);

var formatString = reader.GetNumberFormatString(columnIndex);

if (formatString != null) {

var format = new NumberFormat(formatString);

return format.Format(value, culture);

}

return Convert.ToString(value, culture);

}

See also:

* <https://github.com/andersnm/ExcelNumberFormat>
* <https://www.nuget.org/packages/ExcelNumberFormat>

**Important note when upgrading from ExcelDataReader 2.x**

ExcelDataReader 3 had some breaking changes, and older code may produce error messages similar to:

'IExcelDataReader' does not contain a definition for 'AsDataSet'...

'IExcelDataReader' does not contain a definition for 'IsFirstRowAsColumnNames'...

To fix:

1. Make sure to rename any Excel namespace references in the code to the new namespace ExcelDataReader
2. Make sure the project has a reference to the ExcelDataReader.DataSet package to use AsDataSet()
3. Remove the line of code with IsFirstRowAsColumnNames and change the call to AsDataSet() to something like this:

var result = reader.AsDataSet(new ExcelDataSetConfiguration() {

ConfigureDataTable = (\_) => new ExcelDataTableConfiguration() {

UseHeaderRow = true

}

});

**Important note on .NET Core**

By default, ExcelDataReader throws a NotSupportedException "No data is available for encoding 1252." on .NET Core.

To fix, add a dependency to the package System.Text.Encoding.CodePages and then add code to register the code page provider during application initialization (f.ex in Startup.cs):

System.Text.Encoding.RegisterProvider(System.Text.CodePagesEncodingProvider.Instance);

This is required to parse strings in binary BIFF2-5 Excel documents encoded with DOS-era code pages. These encodings are registered by default in the full .NET Framework, but not on .NET Core.