NPOI の使い方

作成日: 2017/12/12
作成者: 宇野

更新履歴

更新日	更新概要	作業者
2017/12/12	・　新規作成	宇野
	・	
	・	
	・	
	・	

目次

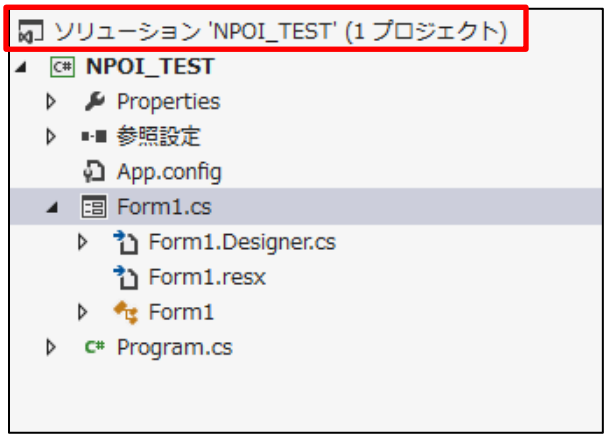
はじめに

NPOI は、Excel の読み込み、書き込みを行う.NET (C#, VB.NET) のライブラリです。COM を使用したときに Excel のプロセスが残ってしまう問題を回避でき、「.xls」、「.xlsx」のどちらにも対応しています。

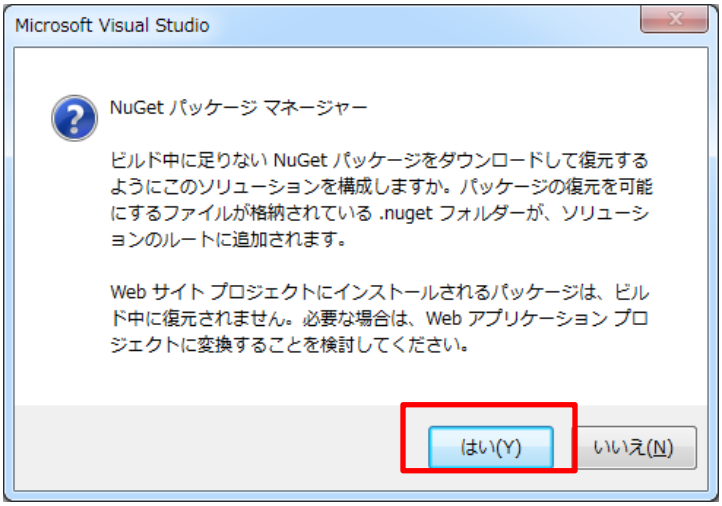
NPOI を使用するための前準備

1. Visual Studio でプロジェクトを作成したのち、赤枠の部分を右クリックして、「NuGet パッケージの復元の有効化」をクリックします。

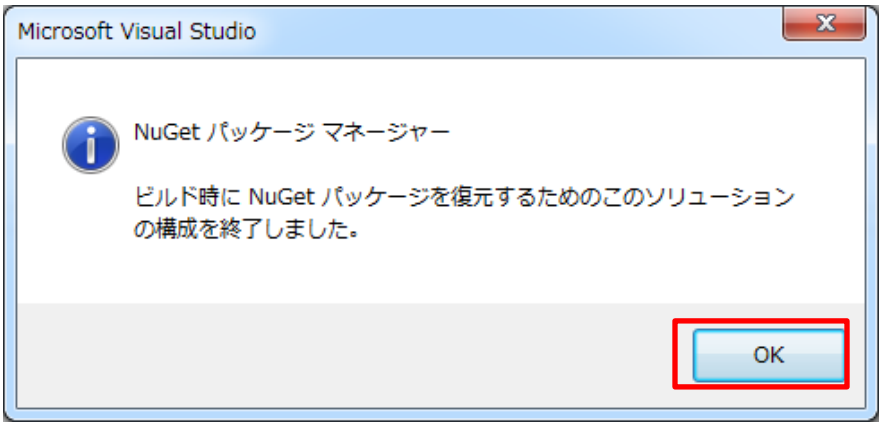
   ※下記は、Visual Studio2012 を使用しています。



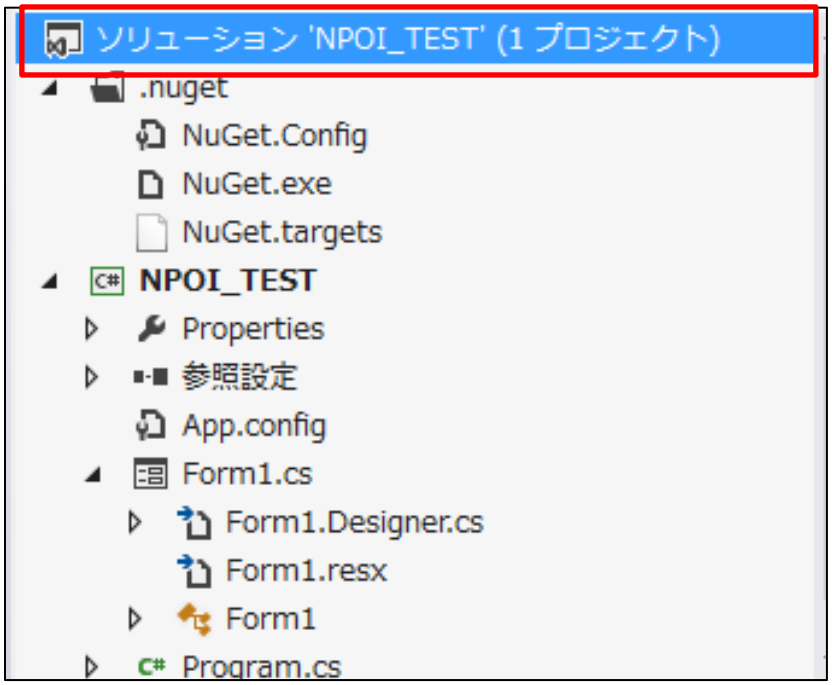
2. 下記のダイアログが出てきたら、「はい」をクリックします。



3.　「2.」のダイアログで「はい」をクリックすると下記のダイアログが表示されます。

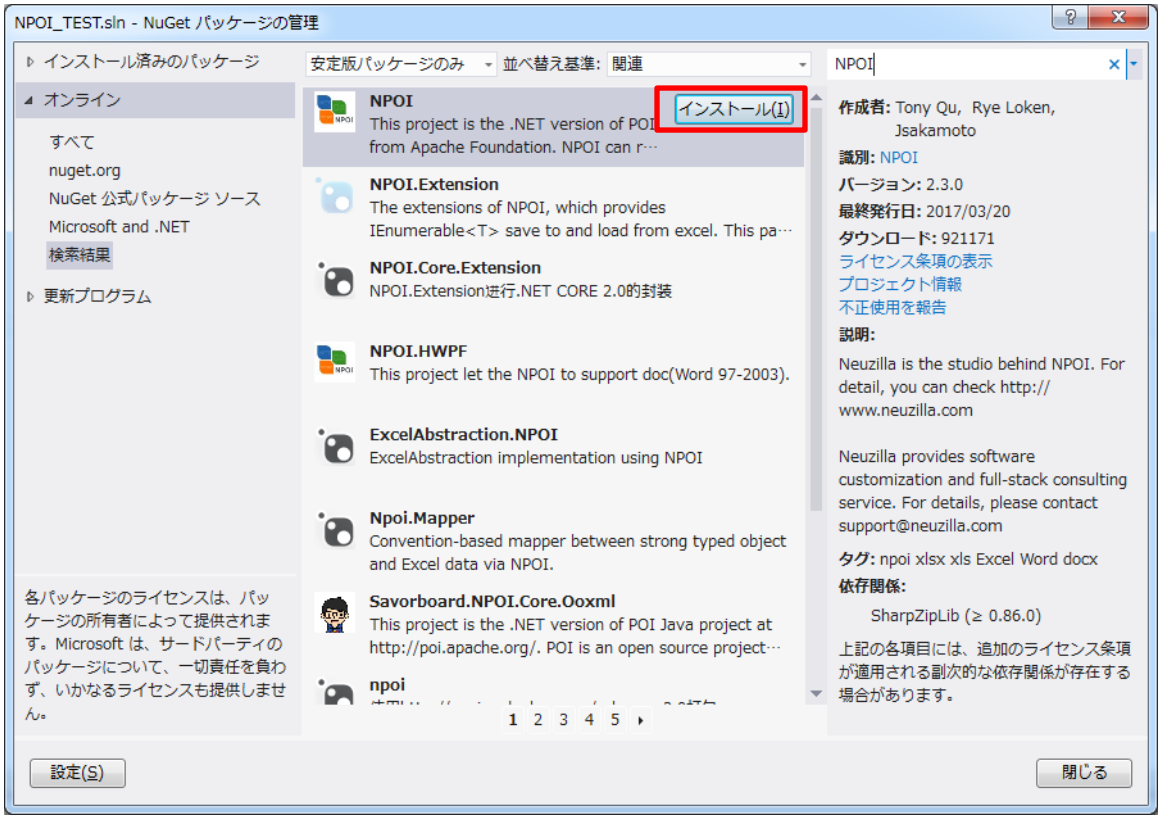


4.　赤枠の部分を右クリックして、「ソリューションの NuGet パッケージの管理」をクリックします。

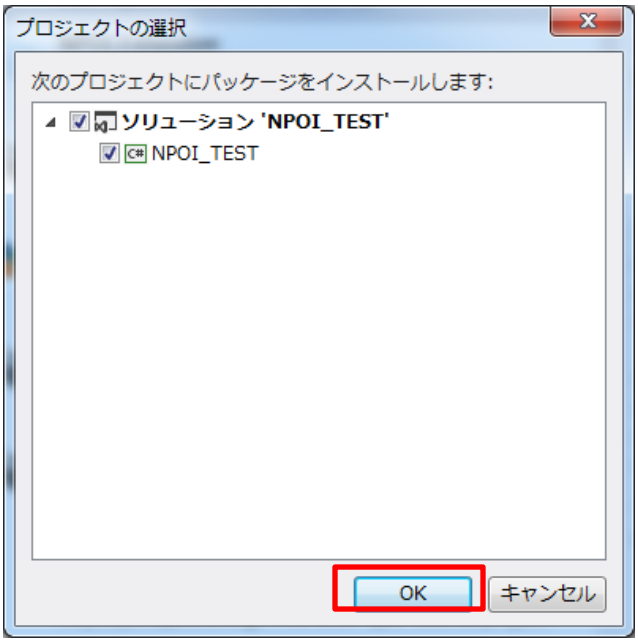


5. NuGet パッケージの管理の画面から NPOI をインストールします。

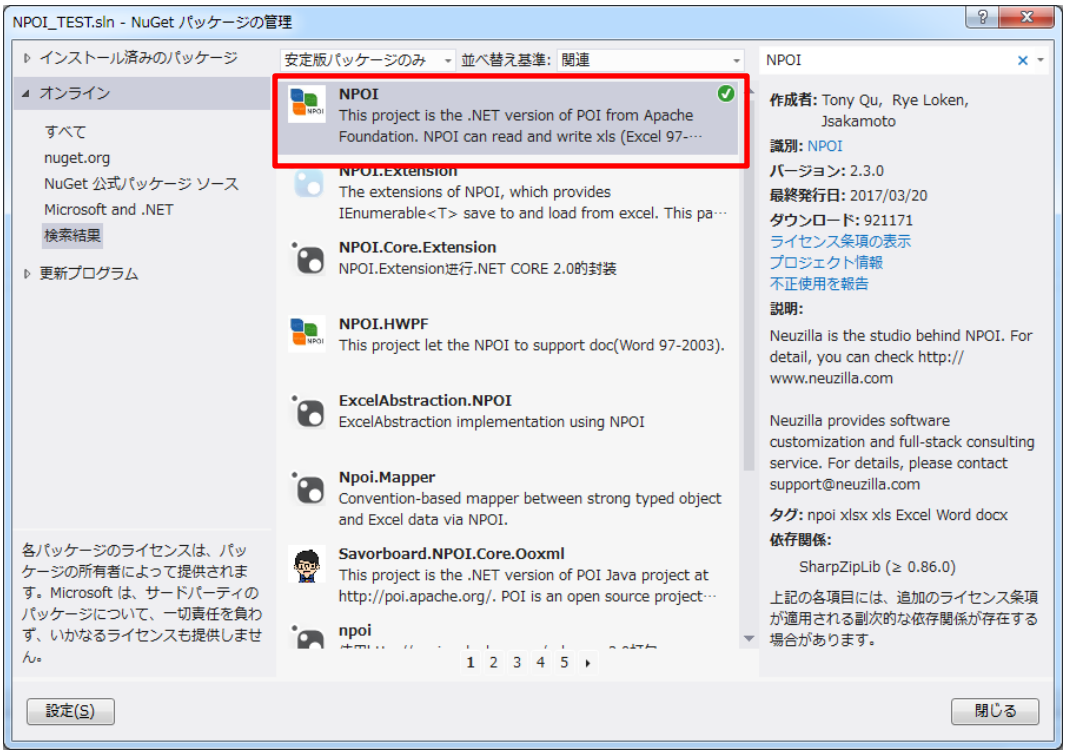
※オンラインをクリックして右上のところで NPOI で検索すれば下記のように「NPOI」が出てきます。



6. 下記の画面で OK をクリックします。



7. NPOI がインストールされると、NuGet パッケージの管理で下記のように表示されます。



NPOI を使用した Excel ファイル読み込み

1. コードの一番上の名前空間(using)に NPOI に関する処理をセットします。

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.IO;
using NPOI;
using NPOI.SS.UserModel;
using NPOI.HSSF.UserModel;
using NPOI.XSSF.UserModel;
```

2. 読み込むブックとシートを定義します。

※「IWorkbook」は、読み込む Excel ファイルです。（path は、読み込む Excel ファイルのパス）

※「ISheet」は、読み込む Excel ファイルのシート

```
private void FileRead(String path)
{
    IWorkbook book = WorkbookFactory.Create(path);
    ISheet sheet = book.GetSheetAt(0);
```

3.　Excel ファイルの値の読み込みを行います。

※下記のコードでは、「GetExcelValue」というメソッドで、Excel ファイルの値を取得しています。

※Excel ファイルの”A1”セルで「(行, 列) = (0, 0)」と考えて取得します。

※下記青枠のところでは、エラー回避を行っています。

(例：rows が null のときに rows.GetCell(col)を実行するとエラーになります。)

※下記黄枠のところでは、Excel のセルの型によって取得する方法が変わっています。

```
while (txt1 != "" || txt2 != "" || txt3 != "" || txt4 != "")
{
    txt1 = GetExcelValue(sheet, 1, 1 + cell);
    txt2 = GetExcelValue(sheet, 2, 1 + cell);
    txt3 = GetExcelValue(sheet, 3, 1 + cell);
    txt4 = GetExcelValue(sheet, 4, 1 + cell);

    text1.Add(txt1);
    text2.Add(txt2);
    text3.Add(txt3);
    text4.Add(txt4);
    cell++;
}
```

```
string value = "";
IRow rows;
ICell cell;
rows = sheet.GetRow(row);
if (rows == null)
{
    value = "";
}
else
{
    cell = rows.GetCell(col);
    if (cell == null)
    {
        value = "";
    }
    else
    {
        switch (cell.CellType)
        {
            case CellType.String:
                value = cell.StringCellValue;
                break;
            case CellType.Numeric:
                if (DateUtil.IsCellDateFormatted(cell))
                {
                    value = cell.DateCellValue.ToString();
                }
                else
                {
                    value = cell.NumericCellValue.ToString();
                }
                break;
            case CellType.Boolean:
                value = cell.BooleanCellValue.ToString();
                break;
            default:
                value = "";
                break;
```

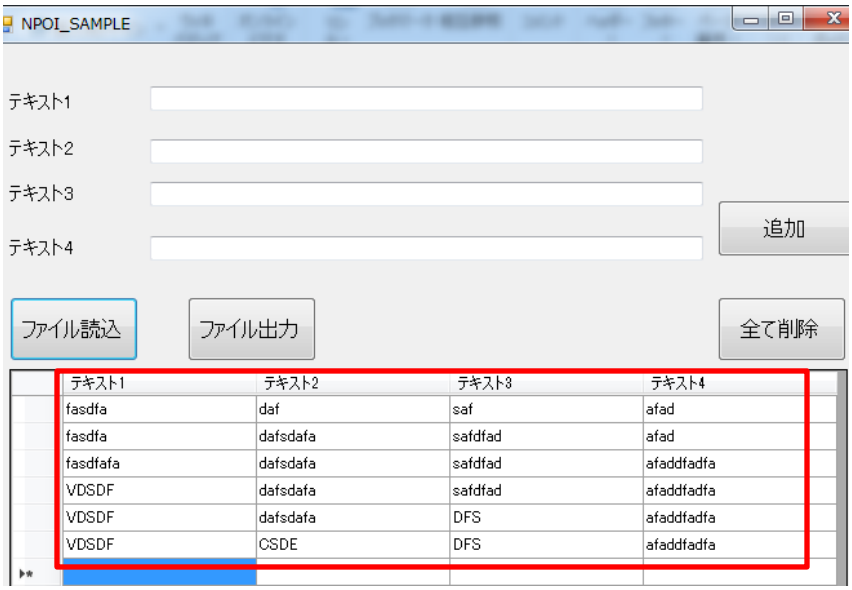
4.　読み込みが完了したら、読み込みに使用したブックを閉じます。

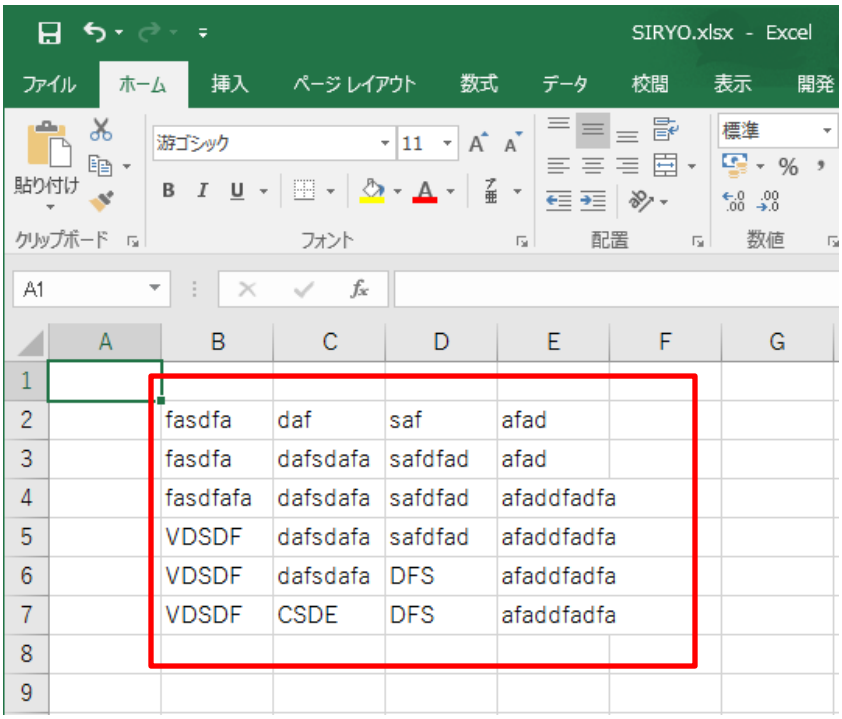
※下記の処理がないと、使用した Excel ファイルが開いたままの状態となります。

```
book.Close();
```

・下記のサンプル(上)では、データグリッドビューに Excel ファイルの値が表示されます。

※Excel ファイルの B2 セルの値がグリッドの左上に表示されています。





NPOI を使用した Excel ファイル書き込み

1. コードの一番上の名前空間(using)に NPOI に関する処理をセットします。

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.IO;
using NPOI;
using NPOI.SS.UserModel;
using NPOI.HSSF.UserModel;
using NPOI.XSSF.UserModel;
```

2. 出力するブックとシート、出力時に必要な FileStream を定義します。

※拡張子が".xlsx"の場合は XSSFWorkbook、".xls"の場合は HSSFWorkbook を使用します。

※path は、出力する Excel ファイルのパスです。

```
XSSFWorkbook bookXSS = new XSSFWorkbook();
HSSFWorkbook bookHSS = new HSSFWorkbook();
ISheet XSSsheet = bookXSS.CreateSheet("sheet");
ISheet HSSsheet = bookHSS.CreateSheet("sheet");
FileStream fs = new FileStream(path, FileMode.Create);
```

3. Excel ファイルへの書き込みを行います。

※下記コードでは、「WriteCell」というメソッドで Excel ファイルへの書き込みを行っています。

※Excel ファイルの A1 セルを「(行,列) = (0, 0)」と考えて、Excel ファイルのセルに書き込みます。

```
private void WriteSet(ISheet sheet)
{

    GridCall();

    for (int i = 0; i < GridHash.Rows.Count - 1; i++)
    {
        WriteCell(sheet, 1, 1 + i, text1[i]);
        WriteCell(sheet, 2, 1 + i, text2[i]);
        WriteCell(sheet, 3, 1 + i, text3[i]);
        WriteCell(sheet, 4, 1 + i, text4[i]);
    }

}
```

```
public static void WriteCell(ISheet sheet, int col, int row, string value)
{
    IRow rows = sheet.GetRow(row) ?? sheet.CreateRow(row);
    ICell cell = rows.GetCell(col) ?? rows.CreateCell(col);

    cell.SetCellValue(value);
}
```

4. Excel ファイルの出力処理を行います。

```
switch (Path.GetExtension(path))
{
    case ".xlsx":
        WriteSet(XSSsheet);
        bookXSS.Write(fs);
        break;
    case ".xls":
        WriteSet(HSSsheet);
        bookHSS.Write(fs);
        break;
}
```

5. 使用したブックを閉じて、FileStream のリソースを開放します。

```
fs.Dispose();

bookXSS.Close();
bookHSS.Close();
```

・下記サンプル（上）では、データグリッドビューの値が Excel ファイルに出力されます。

※グリッドビューの左上の値が Excel ファイルの”B2”セルに出力されます。