

# Excel for WPF/Silverlight

2018.04.11 更新

グレースィティ株式会社


## 目次

<a href="#">製品の概要</a>	2
<a href="#">主な特長</a>	2-3
<a href="#">Excel for WPF クイックスタート</a>	4
<a href="#">手順 1:プロジェクトの設定</a>	4
<a href="#">手順 2:C1XLBook へのコンテンツの追加</a>	4-5
<a href="#">手順 3:コンテンツの書式設定</a>	5-6
<a href="#">手順 4:XLS ファイルを保存して開く</a>	6-7
<a href="#">Excel for Silverlight クイックスタート</a>	8
<a href="#">手順 1:プロジェクトの設定</a>	8
<a href="#">手順 2:C1XLBook へのコンテンツの追加</a>	8-9
<a href="#">手順 3:XLSX ファイルの保存</a>	9-10
<a href="#">手順 4:プログラムの実行</a>	10
<a href="#">Excel for WPF/Silverlight の使い方</a>	11
<a href="#">ドキュメントの作成</a>	11-12
<a href="#">ワークシート</a>	12-13
<a href="#">行と列</a>	13
<a href="#">セル</a>	13
<a href="#">スタイル</a>	13
<a href="#">タスク別ヘルプ</a>	14
<a href="#">セルへのコメントの追加 (WPFのみ)</a>	14-15
<a href="#">ワークブックへのコンテンツの追加</a>	15-16
<a href="#">セルへの画像の追加 (WPFのみ)</a>	16-21
<a href="#">ワークシートへのページブレークの追加</a>	21-22
<a href="#">ブック間での行のコピー (WPFのみ)</a>	22-23
<a href="#">小計の作成</a>	23-25
<a href="#">セルの書式設定</a>	25-26
<a href="#">OpenXml ファイルのインポートとエクスポート (WPFのみ)</a>	26-28
<a href="#">CSV ファイルの保存 (Silverlightのみ)</a>	28-30
<a href="#">ワークブックの計算モードの設定</a>	30-31

## 製品の概要

**Excel for WPF/Silverlight** を使用すると、.NET アプリケーションがなくても Excel® データを簡単に操作できます。Microsoft® Excel がインストールされている必要もありません。Excel 97 以上の XLS ファイルを作成または読み込むことができます。**Excel for WPF/Silverlight** は、新しい Office 2007 OpenXml 形式をサポートします。この形式を使用すると、小さく圧縮された XLSX ファイルを保存できます。

**Excel for WPF/Silverlight** の主要なコンポーネントは、**C1XLBook** オブジェクトです。これは、いくつかのシートを含む Excel ワークブックを表します。**C1XLBook** を使用して、既存の Excel ファイルを読み込んだり、新しい Excel ファイルを作成します。次に、シート、スタイル、ハイパーリンク、画像、ヘッダー/フッター、ページブレイクなどを追加します。完了したら、**C1XLBook** をファイルまたはストリームに保存して終了します。Excel ユーザーであれば誰でもそのデータにアクセスできます。このようにとても簡単です。

 **メモ:** 説明内に含まれるクラスおよびメンバーに対するリファレンスへのリンクは、原則として WPF 版のリファレンスページを参照します。Silverlight 版については、目次から同名のメンバーを参照してください。

## 主な特長

以下に、**Excel for WPF/Silverlight** の便利な機能をいくつか示します。

- **1つのコマンドを使用したワークブックの保存またはロード**  
**Excel for WPF/Silverlight** は使いやすく、1つのコマンドを使用してワークブックをロードまたは保存し、シートをグリッドコントロールのように操作できます。
- **個々のセル内のデータの読み書き**  
**C1XLBook** をロードまたは作成したら、単純なグリッドのように各シート内のデータにアクセスできます。次に例を示します。

XAML

```
XLSheet sheet = C1XLBook.Sheets[0];
sheet[0, 0].Value = DateTime.Now;
```

- **各セルのデータの書式設定**  
各セルに関連付けられた書式には、セルに格納されているデータと同様に簡単にアクセスできます。次に例を示します。

C#

```
XLStyle style = new XLStyle(c1XLBook1);
style.Format = "dd-MM-yyyy";
style.Font = new XLFont("Courier New", 14);
XLSheet sheet = c1XLBook1.Sheets[0];
sheet[0, 0].Value = DateTime.Now;
sheet[0, 0].Style = style;
```

- **Excel for WPF/Silverlight を使用した XLS ファイルのエクスポート**  
他の ComponentOne コンポーネントは **Excel for WPF/Silverlight** を使用して XLS ファイルをエクスポートします。たとえば、**C1Report** は、**Excel for WPF/Silverlight** を使用して、レポートの XLS バージョンを作成します。これで、Microsoft Excel のユーザーであれば誰でも、これらのレポートを表示および編集できます。
- **Microsoft Excel を使用しない .xls および .xlsx ファイルの読み書き**  
**Excel for WPF/Silverlight** は、.xls (Excel 97 以降) および .xlsx (OpenXml 形式) ファイルを読み書きします。後者のファイルを再利用し、ファイルをやり取りしたり、圧縮してファイルサイズを小さくすることができます。Microsoft Excel がインストールされている必要はありません。

- **セル内の画像の作成および配置**

セルに画像を追加できるだけでなく、セルのサイズやセル内の画像の位置を指定できます。また、画像をセルに合わせて拡大縮小したり、クリップしたり、引き延ばすかどうかを指定できます。

- **ストリームへのファイルの保存および読み込み**

**Load** メソッドと **Save** メソッドの新しいオーバーロードメソッドを使用して、ワークブックをメモリストリームから直接読み込んだり、メモリストリームに直接書き込めるようになりました。一時ファイルを使用する必要はなくなりました。

- **シートのヘッダーおよびフッターへの画像の追加**

**XLPrintSettings** クラスのプロパティを使用して、シートのヘッダーまたはフッターの左、中央、または右部分に画像を追加できます。

## Excel for WPF クイックスタート

このクイックスタートでは、**Excel for WPF** のいくつかの機能について詳しく説明します。このクイックスタートでは、ワークブックを作成し、そのワークブックに書式設定されたデータを追加し、XLS ファイルを保存してから開きます。

### 手順 1: プロジェクトの設定

この手順では、フォームに C1.WPF.Excel.4.dll への参照を追加します。

1. 新しい WPF プロジェクトを作成します。
2. プロジェクトの[プロジェクト]メニューから[参照の追加]オプションを選択します。[参照の追加]ダイアログボックスが表示されます。
3. **C1.WPF.Excel.4.dll** を参照して選択し、[OK]をクリックします。
4. フォームをダブルクリックして **Window\_Loaded** イベントを追加し、コードビューに切り替えます。Imports (Visual Basic) または **using** (C#) 文をフォームの先頭のコードに追加します。これで、C1.WPF.Excel 名前空間内のすべての名前を使用できます。

#### VisualBasic

```
Imports C1.WPF.C1Excel
```

#### C#

```
using C1.WPF.Excel;
```

次に、コンテンツを作成して **C1XLBook** に追加します。

### 手順 2: C1XLBook へのコンテンツの追加

Visual Studio プロジェクトでコードビューを表示したまま、次のコードを追加して、ブックとそのコンテンツを作成します。

#### VisualBasic

```
Dim book As New C1XLBook()  
Dim i As Integer  
    Dim sheet As XLSheet = book.Sheets(0)  
    For i = 0 To 9  
        sheet(i, 0).Value = (i + 1) * 10  
        sheet(i, 1).Value = (i + 1) * 100  
        sheet(i, 2).Value = (i + 1) * 1000  
    Next  
End Function  
End Sub
```

## C#

```
C1XLBook book = new C1XLBook();
int i;
XLSheet sheet = book.Sheets[0];
for (i = 0; i <= 9; i++)
{
    sheet[i, 0].Value = (i + 1) * 10;
    sheet[i, 1].Value = (i + 1) * 100;
    sheet[i, 2].Value = (i + 1) * 1000;
}
```

## 手順 3:コンテンツの書式設定

次に、スタイルを使用してコンテンツを書式設定します。この手順のコードは、「[手順 2:C1XLBook へのコンテンツの追加](#)」のコードの後に追加する必要があります。

1. 次のコードを追加して、2つの新しいスタイル(style1 と style2)を作成します。

### VisualBasic

```
' スタイル1を追加します。
Dim style1 As New XLStyle(book)
style1.Font = New XLFont("Tahoma", 9, true, false)
style1.ForeColor = Colors.RoyalBlue
' スタイル2を追加します。
Dim style2 As New XLStyle(book)
style2.Font = New XLFont("Tahoma", 9, false, true)
style2.BackColor = Colors.RoyalBlue
style2.ForeColor = Colors.White
```

## C#

```
// スタイル1を追加します。
XLStyle style1 = new XLStyle(book);
style1.Font = new XLFont("Tahoma", 9, true, false);
style1.ForeColor = Colors.RoyalBlue;
// スタイル2を追加します。
XLStyle style2 = new XLStyle(book);
style2.Font = new XLFont("Tahoma", 9, false, true);
style2.BackColor = Colors.RoyalBlue;
style2.ForeColor = Colors.White;
```

2. 次に、次のコードを追加して、コンテンツに新しいスタイルを適用します。

### VisualBasic

```

For i = 0 To 9
    ' コンテンツにスタイルを適用します。
    If (i + 1) Mod 2 = 0 Then
        sheet(i, 0).Style = style2
        sheet(i, 1).Style = style1
        sheet(i, 2).Style = style2
    Else
        sheet(i, 0).Style = style1
        sheet(i, 1).Style = style2
        sheet(i, 2).Style = style1
    End If
Next i

```

## C#

```

for (i = 0; i <= 9; i++)
{
    // コンテンツにスタイルを適用します。
    if ((i + 1) % 2 == 0)
    {
        sheet[i, 0].Style = style2;
        sheet[i, 1].Style = style1;
        sheet[i, 2].Style = style2;
    }
    else
    {
        sheet[i, 0].Style = style1;
        sheet[i, 1].Style = style2;
        sheet[i, 2].Style = style1;
    }
}

```

## 手順 4: XLS ファイルを保存して開く

最後に、次のコードを追加して、Excel ワークブックを保存してからロードします。このコードは、**Form\_Load** イベント内の「[手順 3: コンテンツの書式設定](#)」のコードの後に追加する必要があります。

### VisualBasic

```

ClXlBook1.Save("c:\mybook.xls")
System.Diagnostics.Process.Start("C:\mybook.xls")

```

## C#

```

clXlBook1.Save(@"c:\mybook.xls");
System.Diagnostics.Process.Start(@"c:\mybook.xls");

```

## プログラムの実行と動作の確認

	A	B	C
1	10	100	1000
2	20	200	2000
3	30	300	3000
4	40	400	4000
5	50	500	5000
6	60	600	6000
7	70	700	7000
8	80	800	8000
9	90	900	9000
10	100	1000	10000

書式設定されたコンテンツがワークブックに追加されました。

おめでとうございます。これで、**Excel for WPF** クイックスタートは終了です。



## Excel for Silverlight クイックスタート

このクイックスタートでは、**Excel for Silverlight**の一部の機能について詳しく説明します。このクイックスタートでは、プロジェクトに **C1XLBook** を追加し、ワークブックに書式設定されたデータを追加し、XLSファイルを保存してから開きます。

### 手順 1: プロジェクトの設定

この手順では、新しいプロジェクトを作成し、C1.Silverlight.Excel アセンブリへの参照を追加します。

1. 新しいSilverlight プロジェクトを作成します。
2. **[プロジェクト]**→**[参照の追加]**を選択し、C1.Silverlight.Excel.5.dllを参照して見つけます。**[OK]**をクリックします。
3. ページに標準のボタンコントロールを追加し、その **Content** プロパティを **保存** に設定します。
4. ボタンをダブルクリックして、MainPage.xaml.csのコードビューに切り替えます。これにより、コードに **button1\_Click** イベントも追加されます。
5. **Imports** (Visual Basic) または **using** (C#) 文をフォームの先頭のコードに追加します。これで、C1.Silverlight.Excel名前空間内のすべての名前を使用できます。

#### VisualBasic

```
Imports C1.Silverlight.Excel
```

#### C#

```
using C1.Silverlight.Excel;
```

これで、**C1XLBook** にコンテンツを追加できます。

### 手順 2: C1XLBook へのコンテンツの追加

手順 1 で作成した **button1\_Click** イベントで、次のコードを追加して、ブックとそのコンテンツを作成します。コードは次のようになります。

#### VisualBasic

```
Private Sub button1_Click(sender As Object, e As RoutedEventArgs)
    SaveBook(Function(book)
        Dim i As Integer
        Dim sheet As C1.Silverlight.Excel.XLSheet = book.Sheets(0)
        For i = 0 To 9
            sheet(i, 0).Value = (i + 1) * 10
            sheet(i, 1).Value = (i + 1) * 100
            sheet(i, 2).Value = (i + 1) * 1000
        Next
    End Function)
```

```
End Sub
```

## C#

```
private void Application_Startup(object sender, StartupEventArgs e)
{
    SaveBook(book =>
        {
            int i;
            Cl.Silverlight.Excel.XLSheet sheet = book.Sheets[0];
            for (i = 0; i <= 9; i++)
            {
                sheet[i, 0].Value = (i + 1) * 10;
                sheet[i, 0].Value = (i + 1) * 100;
                sheet[i, 0].Value = (i + 1) * 1000;
            }
        });
}
```

**SaveBook**メソッドを呼び出していることに注目してください。次の手順では、このメソッドのコードを追加します。

## 手順 3: XLSX ファイルの保存

次のコードを追加して、Excel ワークブックを保存します。アプリケーションを実行すると、このコードによって**[名前を付けて保存]**ダイアログボックスが開きます。このダイアログボックスで、.xlsx ファイルを任意の場所に保存できます。

## VisualBasic

```
Private Sub SaveBook(action As Action(Of ClXLBook))
Dim dlg = New SaveFileDialog()
dlg.Filter = "Excel Files (*.xlsx)|*.xlsx"
If dlg.ShowDialog() = True Then
    Try
        Dim book = New ClXLBook()
        action(book)
        Using stream = dlg.OpenFile()
            book.Save(stream)
        End Using
    Catch x As Exception
        MessageBox.Show(x.Message)
    End Try
End If
End Sub
```

## C#

```
private void SaveBook(Action<ClXLBook> action)
{
```

```

var dlg = new SaveFileDialog();
dlg.Filter = "Excel Files (*.xlsx)|*.xlsx";
if (dlg.ShowDialog() == true)
{
    try
    {
        var book = new ClXLBook();
        if (action != null)
        {
            action(book);
        }
        using (var stream = dlg.OpenFile())
        {
            book.Save(stream);
        }
    }
    catch (Exception x)
    {
        MessageBox.Show(x.Message);
    }
}
}

```

## 手順 4:プログラムの実行

[F5]キーを押してアプリケーションを実行します。

1. **[保存]**ボタンをクリックします。**[名前を付けて保存]**ダイアログボックスが表示されます。
2. ワークブックのファイル名を入力し、**[保存]**をクリックします。
3. ブックを開きます。次の画像のように表示されます。

	A	B	C
1	10	100	1000
2	20	200	2000
3	30	300	3000
4	40	400	4000
5	50	500	5000
6	60	600	6000
7	70	700	7000
8	80	800	8000
9	90	900	9000
10	100	1000	10000
11			

おめでとうございます。これで、**Excel for Silverlight**クイックスタートは終了です。

## Excel for WPF/Silverlight の使い方

以下のトピックでは、XLS ファイルの作成方法について説明すると共に、ファイルを構成するコンポーネント(ワークシート、行、列、セル、スタイル)の作成に使用される **Excel for WPF/Silverlight** の主要なクラスについて説明します。

### ドキュメントの作成

**Excel for WPF/Silverlight** を使用して新しい XLS ファイルを作成するには、次の3つの手順を実行する必要があります。

1. C1.WPF.Excel.4.dll/C1.Silverlight.Excel.5.dll への参照を追加し、**C1XLBook** を作成します。
2. コンテンツをシートに追加します。各シートには、**Value** プロパティと **Style** プロパティを持ついくつかのセル(**XLCell** オブジェクト)が格納されます。
3. **Save** メソッドを使用して、ブックをファイルに保存します。

たとえば、次のコードは、1～100 までの数値を含む1枚のシートから成る新しい Excel ファイルを作成します。

```
C#  
  
// 手順 1: 保存する新規ブックを作成します  
C1XLBook book = new C1XLBook();  
// 手順 2: いくつかのセルの中身を書き込みます  
XLSheet sheet = book.Sheets[0];  
    for (int i = 0; i < 100; i++)  
        sheet[i, 0].Value = i + 1;  
// 手順 3: ファイルを保存して開きます  
book.Save(@"C:\MyFiles\Test\test.xls");  
System.Diagnostics.Process.Start(@"C:\MyFiles\Test\test.xls");
```

手順 2 が最も興味深い作業です。このコードは、最初に、新しい Excel ブックに1枚のワークシートを表示する **XLSheet** オブジェクトを取得します。このシートは、新しい **C1XLBook** を追加または作成する際に自動的に作成されます。次に、シートインデクサを使用してシート内のセルを参照し、セルに1～100 までの値を割り当てます。

**XLSheet** オブジェクト内のインデクサは、必要に応じてセルを自動的に作成します。これで、作成するワークシートに簡単に値を入力できます。シートのサイズを調べる場合は、シートの **Rows.Count** プロパティと **Columns.Count** プロパティを使用します。

もちろん、セルに値を割り当てることのできるだけではありません。スタイルを使用して、セルの書式を設定することもできます。それには、1つ以上の **XLStyle** オブジェクトを作成し、そのオブジェクトを値の割り当てと同じ手順でセルに割り当てます。次に示すこのコードの改訂版では、シートを作成し、偶数の数値を太字の赤色の文字と黄色の背景色で強調表示し、奇数を青の斜体で示します。

```
C#  
  
// 手順 1: 保存する新規ブックを作成します  
C1XLBook book = new C1XLBook();  
var sheet = book.Sheets[0];  
// 手順 2: 奇数と偶数のスタイルを作成します  
var styleOdd = new XLStyle(book);  
styleOdd.Font = new XLFont("Tahoma", 9, false, true);  
styleOdd.ForeColor = Colors.Blue;  
var styleEven = new XLStyle(book);  
styleEven.Font = new XLFont("Tahoma", 9, true, false);  
styleEven.ForeColor = Colors.Red;
```

```

styleEven.BackColor = Colors.Yellow;
// 手順 3:いくつかのセルの中身を書き込みます
    for (int i = 0; i < 100; i++)
    {
        XLCell cell = sheet[i, 0];
        cell.Value = i + 1;
        cell.Style = ((i + 1) % 2 == 0) ? styleEven : styleOdd;
    }
// 手順 4:ファイルを保存して開きます
book.Save(@"C:\MyFiles\Test\test.xls");
    System.Diagnostics.Process.Start(@"C:\MyFiles\Test\test.xls");

```

コードはほぼ同じです。主な違いは、新しく追加された手順 2です。ここで、奇数のセルと偶数のセルのスタイルが作成されます。手順 3で、セルに新しいスタイルと値が割り当てられます。

上記のコードで作成されたファイルを Microsoft Excel で開くと、次の図のようになります。

	A	B
1	1	
2	2	
3	3	
4	4	
5	5	
6	6	
7	7	
8	8	
9	9	
10	10	
11	11	
12	12	
13	13	
14	14	
15	15	
16	16	
17	17	
18	18	

## ワークシート

ワークシートは、Excel ファイルに含まれる個々のグリッドです。ワークシートは **XLSheet** オブジェクトで表され、これには **C1XLBook** クラスの **Sheets** プロパティからアクセスできます。各シートには名前があり、一連の行と列から成ります。各セルには、行インデックスと列インデックスを取る **XLSheet** インデクサを使用してアクセスできます。

**XLSheet** オブジェクトの **Rows** コレクションと **Columns** コレクションは、それぞれのインデクサを使用する際に自動的に拡張されます。たとえば、次のコードを記述した場合、このシートに 1001 行まで行がなければ、いくつかの新しい行が自動的に追加され、有効な行が返されます。**XLColumn** と **XLCell** のインデクサも同様です。これは、.NET の大部分のコレクションインデクサの動作とは異なりますが、こうすることで、**XLSheet** オブジェクトを簡単に作成し、値を入力できるようになります。

## VisualBasic

```

Dim sheet As XLSheet = C1XLBook1.Sheets(0)
Dim row As XLRow = sheet.Rows(1000)

```

## C#

```
XLSheet sheet = clXLBook1.Sheets[0];  
XLRow row = sheet.Rows[1000];
```

## 行と列

**XLSheet** オブジェクトには、シート内の個別の行と列を公開する行と列のコレクションが含まれています。公開された **XLRow** オブジェクトと **XLColumn** オブジェクトを使用して、シート内の各行と列のサイズ(列の幅と行の高さ)、表示/非表示、およびスタイルを割り当てることができます。これらの値を割り当てない場合は、シートのデフォルト値が使用されます(**DefaultRowHeight** プロパティと **DefaultColumnWidth** プロパティを参照)。

**XLRow** オブジェクトと **XLColumn** オブジェクトのデフォルトのサイズは -1 です。この場合は、シートのデフォルト値が使用されます。

## セル

**XLSheet** オブジェクトにはセルも含まれています。セルには、行インデックスと列インデックスを取るインデクサを使用してアクセスできます。セルは、セル値とスタイルを持つ **XLCell** オブジェクトで表されます。

行と列の場合と同様に、セルインデクサもシートを自動的に拡張します。たとえば、次のように記述したとします。

## VisualBasic

```
Dim cell As XLCell = sheet(10, 10)
```

## C#

```
XLCell cell = sheet[10,10];
```

シートに含まれる行と列の数がそれぞれ 11 未満の場合は、いくつかの行と列が追加され、有効な **XLCell** オブジェクトが返されます。

シートは自動的に拡張されるため、このインデクサが **null** 参照を返すことはありません。シートに特定のセルが存在するかどうかを確認したいが、不要なセルを作成しないようにするには、インデクサの代わりにシートの **GetCell** メソッドを使用します。

**XLCell** オブジェクトには、セルのコンテンツを格納する **Value** プロパティがあります。このプロパティは **object** 型で、文字列、数値、ブール値、DateTime、または null オブジェクトを格納できます。その他の型のオブジェクトは、Excel ファイルに保存できません。

また、**XLCell** オブジェクトには、セルの外観を定義する **Style** プロパティがあります。**Style** プロパティを **null** に設定した場合、セルはデフォルトのスタイルで表示されます。それ以外の場合は、このプロパティにセルの外観(フォント、配置、色、書式設定など)を定義する **XLStyle** オブジェクトを設定する必要があります。

## スタイル

**XLStyle** クラスは、シート内のセル、行、または列の外観を定義します。**XLStyle** には、セル値の表示に使用されるフォント、配置、色、書式設定などのスタイル要素を指定するプロパティが含まれます。すべての **XLStyle** オブジェクトですべてのスタイル要素を定義する必要はありません。たとえば、**XLStyle** で書式設定のみを指定した場合、セルは指定された書式設定で表示されますが、他のスタイル要素(フォント、配置など)にはデフォルト設定が適用されます。

## タスク別ヘルプ

タスク別ヘルプは、Visual Studio プログラミングにある程度慣れていることを前提としています。[ヘルプ]に示される手順に従って作業を進めるだけで、**Excel for WPF/Silverlight** のさまざまな機能を具体的に紹介するプロジェクトを作成しながら、**Excel for WPF/Silverlight** の主要な機能と特長を理解できます。

また、タスク別ヘルプトピックは、新しい .NET プロジェクトが既に作成されており、コードに適切なディレクティブ (C# の場合は using C1.C1.Excel;、Visual Basic の場合は Imports C1.C1.Excel) が追加されているとします。

## セルへのコメントの追加 (WPFのみ)

セルにコメントを追加するには、次の手順に従います。

1. C1.WPF.Excel.4.dll への参照を追加し、**C1XLBook** を作成します。

XAML

```
// 新しいワークブックを作成します
C1XLBook book = new C1XLBook();
```

2. 次のコードを使用して、ワークシート内のセルにテキストを追加します。

XAML

```
book.Sheets[0][2, 3].Value = "テスト";
```

3. 次のコードを使用して、**XLCommentCollection** にコメントを追加し、コメントを表示するボックスを作成します。

XAML

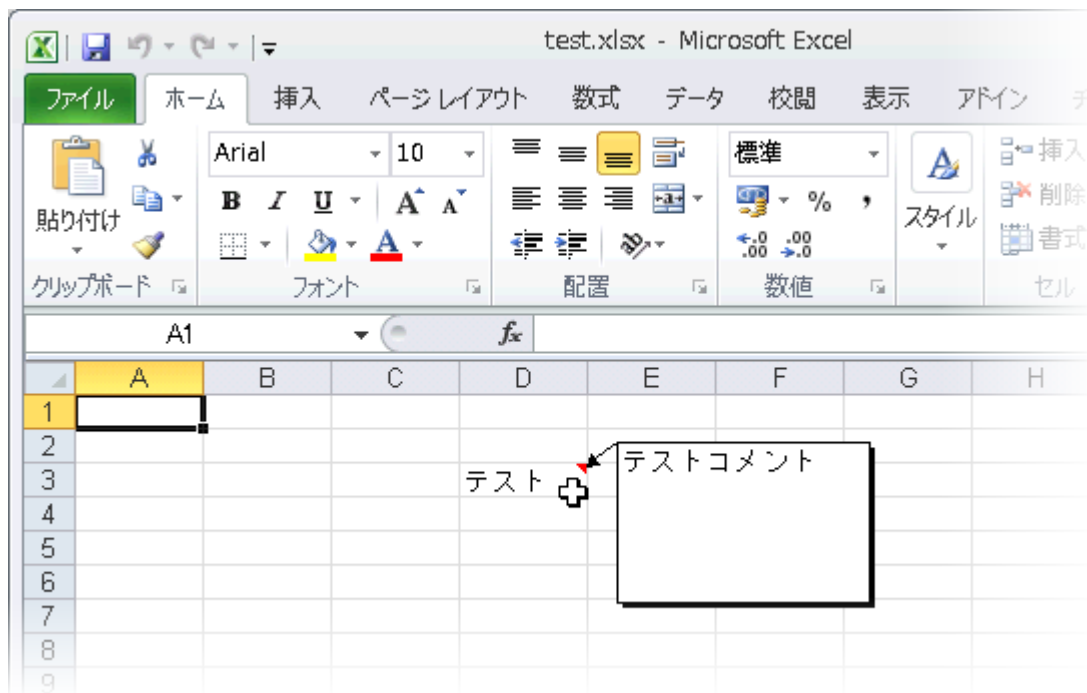
```
book.Sheets[0].Comments.Add(2, 3, "田中", "テストコメント");
book.Sheets[0].Comments[0].TextBox.Rectangle = new Rect(220, 210, 1900, 1200);
```


4. ブックを保存してから開きます。

C#

```
// ファイルを保存してから開きます
book.Save(@"C:\C1Excel\test.xlsx");
System.Diagnostics.Process.Start(@"C:\C1Excel\test.xlsx");
```

5. プログラムを実行します。スプレッドシートが開かれて、次のように表示されます。



 このピックの内容は、ComponentOne for WPF にのみ適用されます。

## ワークブックへのコンテンツの追加

新しいワークブックを作成し、最初の 10 個のセルに値を追加するには、次の手順に従います。

1. C1.WPF.Excel.4.dll/C1.Silverlight.Excel.5.dll への参照を追加し、**C1XLBook** を作成します。

C#

```
// 保存する新しいワークブックを作成します
C1XLBook book = new C1XLBook();
```

2. 最初の 10 個のセルに値を追加します。

C#

```
// 最初の 10 個のセルにコンテンツを書き込みます
XLSheet sheet = book.Sheets[0];

for (int i = 0; i <= 9; i++)
{
    sheet[i, 0].Value = i + 1;
}
```

3. ワークブックを保存してから開きます。このコードは次のようになります。

C#

```
// ファイルを保存してから開きます
book.Save(@"C:\C1Excel\test.xlsx");
System.Diagnostics.Process.Start(@"C:\C1Excel\test.xlsx");
```



	A
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	10

## セルへの画像の追加 (WPFのみ)

次のいずれかの方法を使用して、シートまたはセルに画像を追加できます。このタスクを実行する詳しい手順については、次のリンクをクリックしてください。

### 方法 1: セルの `XLCell.Value` プロパティに画像を直接割り当てる。

この方法を使用すると、画像は元のサイズのままシートに追加されます。画像の左上隅と指定されたセルの左上隅が合わせられます。

1. 既存のワークブックをロードするか、新しいワークブックにいくつかコンテンツを追加します。

### VisualBasic

```
Dim wb As New C1XLBook
wb.Load("C:\Project\WorkBook1.xls")
```

### C#

```
C1XLBook wb = new C1XLBook();
wb.Load(@"C:\Project\WorkBook1.xls");
```

2. 画像を指定し、それをセルの `XLCell.Value` プロパティに割り当てます。

### VisualBasic

```
Dim img As WriteableBitmap
= new WriteableBitmap(new BitmapImage(new Uri("MyImage.bmp",
UriKind.Relative)));
Dim sheet As XLSheet = wb.Sheets("Forecasting Report")
sheet(0, 0).Value = img
```

### C#

# Excel for WPF/Silverlight

```
WriteableBitmap img = new WriteableBitmap(new BitmapImage(new Uri("MyImage.bmp",  
UriKind.Relative))); XLSheet sheet = wb.Sheets["Forecasting Report"];  
sheet[0,0].Value = img;
```

3. 新しいブックを保存してから開きます。


## VisualBasic

```
wb.Save("C:\Project\WorkBook1.xls ")  
System.Diagnostics.Process.Start("C:\Project\WorkBook1.xls")
```

## C#

```
wb.Save(@"C:\Project\WorkBook1.xls");  
System.Diagnostics.Process.Start(@"C:\Project\WorkBook1.xls");
```

この例では、最初のセルの値が画像で置き換えられます。画像は、最初のセルに元のサイズで表示されます。

	A	B	C	D
1			プロジェクトマネージャー: ジョーンズミス	
2			完了日:	2007年1月10日
3				
4	スケジュール及びコスト			
5	総予算(BAC)	\$2,000		プロジェクトが完
6	出来高実績値(EV)	\$1,050		現時点までに完
7	コスト実績値(AC)	\$950		実際の工数であ
8	出来高計画値(PV)	\$10,000		各フェーズのスケ
9				

**方法 2: XLPictureShape オブジェクトを作成し、そのプロパティを設定してから、セルの XLCell.Value プロパティに割り当てる。**

2番目の方法では、サイズ、回転角度、明度、コントラスト、境界線などを指定して画像をカスタマイズできます。

1. 既存のワークブックをロードするか、新しいワークブックにいくつかコンテンツを追加します。

## VisualBasic

```
Dim wb As New C1XLBook  
wb.Load("C:\Project\WorkBook1.xls")
```

## C#

```
C1XLBook wb = new C1XLBook();  
wb.Load(@"C:\Project\WorkBook1.xls");
```

2. **XLPictureShape** オブジェクトを作成し、いくつかのプロパティを設定してから、セルの `XLCell.Value` プロパティに割り当てます。

## VisualBasic

```
Dim img As WriteableBitmap
    = new WriteableBitmap(new BitmapImage(new Uri("MyImage.bmp",
UriKind.Relative)));
Dim pic As New XLPictureShape(img, 1500, 1500)
pic.Rotation = 30.0F
pic.LineColor = Color.DarkRed
pic.LineWidth = 100
' 指定されたシートの最初のセルに画像を割り当てます
Dim sheet As XLSheet = wb.Sheets("Forecasting Report")
sheet(0, 0).Value = pic
```

## C#

```
WriteableBitmap img = new WriteableBitmap(new BitmapImage(new Uri("MyImage.bmp",
UriKind.Relative)));
XLPictureShape pic = new XLPictureShape(img, 1500, 1500);
pic.Rotation = 30.0f;
pic.LineColor = Color.DarkRed;
pic.LineWidth = 100;
// 指定されたシートの最初のセルに画像を割り当てます
XLSheet sheet = wb.Sheets("Forecasting Report");
sheet[0,0].Value = pic;
```

3. ブックを保存してから開きます。

## VisualBasic

```
wb.Save("C:\Project\WorkBook1.xls ")
System.Diagnostics.Process.Start("C:\Project\WorkBook1.xls")
```


## C#

```
wb.Save(@"C:\Project\WorkBook1.xls");
System.Diagnostics.Process.Start(@"C:\Project\WorkBook1.xls");
```

この例では、最初のセルの値が画像で置き換えられます。画像は、30°回転し、濃い赤色の境界線で囲まれます。ここでは、画像の水平および垂直位置を指定したので、画像は最初のセルには表示されません。

# Excel for WPF/Silverlight

	A	B	C
1			プロジェクトマネジャー:
2	部門: 開発部		完了日:
3			
4	スケジュール及び		
5	総予算(BAO)	\$2,000	
6	出来高実績値(EV)	\$1,050	
7	コスト実績値(AC)	950	
8	出来高計画値(PV)	100	
9			
10			
11			
12	マイルストーン		



方法 3: XLPictureShape オブジェクトを作成し、そのプロパティを設定してから、シートの ShapeCollection に追加する。

この方法では、XLPictureShape コンストラクタを使用して、シート座標で画像の境界を指定します。この形状は、特定のセルではなく、シートの ShapeCollection に直接追加されます。

1. 既存のワークブックをロードするか、新しいワークブックにいくつかコンテンツを追加します。

## VisualBasic

```
Dim wb As New C1XLBook  
wb.Load("C:\Project\WorkBook1.xls")
```

## C#

```
C1XLBook wb = new C1XLBook();  
wb.Load(@"C:\Project\WorkBook1.xls");
```

2. XLPictureShape オブジェクトを作成し、いくつかのプロパティを設定してから、シートの ShapeCollection に割り当てます。

## VisualBasic

```
Dim img As WriteableBitmap  
= new WriteableBitmap(new BitmapImage(new Uri("MyImage.bmp",  
UriKind.Relative)));  
Dim pic As New XLPictureShape(img, 3000, 3500, 2500, 900)  
pic.Rotation = 30.0F  
pic.LineColor = Color.DarkRed  
pic.LineWidth = 100  
' 指定されたシートの ShapeCollection に画像を追加します  
Dim sheet As XLSheet = wb.Sheets("Forecasting Report")  
sheet.Shapes.Add(pic)
```

## C#

```
WriteableBitmap img = new WriteableBitmap(new BitmapImage(new Uri("MyImage.bmp",
UriKind.Relative)));
XLPictureShape pic = new XLPictureShape(img, 3000, 3500, 2500, 900);
pic.Rotation = 30.0f;
pic.LineColor = Color.DarkRed;
pic.LineWidth = 100;
// 指定されたシートの ShapeCollection に画像を追加します
XLSheet sheet = wb.Sheets("Forecasting Report");
sheet.Shapes.Add(pic);
```

3. ブックを保存してから開きます。

## VisualBasic

```
wb.Save("C:\Project\WorkBook1.xls ")
System.Diagnostics.Process.Start("C:\Project\WorkBook1.xls")
```


## C#


```
wb.Save(@"C:\Project\WorkBook1.xls");
System.Diagnostics.Process.Start(@"C:\Project\WorkBook1.xls");
```

この例では、形状がシートの ShapeCollection に追加されたため、最初のセルの値は画像で置き換えられません。ここでは、画像の高さと幅を指定し、水平および垂直位置を指定しました。

# Excel for WPF/Silverlight

	A	B	C
1	プロジェクト名: Project 3X		プロジェクトマ:
2	部門: 開発部		完了日:
3			
4	スケジュール及びコスト		
5	総予算(BAC)	\$2,000	
6	出来高実績値(EV)	\$1,050	
7	コスト実績値(AC)	\$950	
8	出来高計画値(PV)	\$10,000	
9			
10			
11			
12	マイルストーン		
13	マイルストーン		
14	マイルストーンA: 契約書を締結		
15	マイルストーンB: ビジネス要件を設定		
16	マイルストーンC: 機能仕様を設定		
17	マイルストーンD: フェーズゲートをレビュー		
18	範囲		



 このピックの内容は、ComponentOne for WPF にのみ適用されます。

## ワークシートへのページブレイクの追加

**PageBreak** プロパティと **PageBreak** プロパティを使用して、OpenXML (.xlsx) 形式のファイルの行および列にページブレイクを簡単に追加できます。

1. C1.WPF.Excel.4.dll/C1.Silverlight.Excel.5.dll への参照を追加し、**C1XLBook** を作成します。

```
C#  
  
// 保存する新しいワークブックを作成します  
C1XLBook book = new C1XLBook();
```

2. 次のコードを使用して、いくつかのテキスト値とページブレイクを追加します。

```
C#  
  
book.Sheets[0][2, 3].Value = "ページ1";  
book.Sheets[0].Rows[2].PageBreak = true;  
  
book.Sheets[0][0, 1].Value = "テスト1";  
book.Sheets[0][0, 2].Value = "テスト2";  
  
book.Sheets[0].Columns[1].PageBreak = true;  
book.Sheets[0][3, 3].Value = "ページ2";
```

3. .xlsx ファイルを保存してから開きます。

```
C#
```

```
// ファイルを保存してから開きます
book.Save(@"C:\C1Excel\test.xlsx");
System.Diagnostics.Process.Start(@"C:\C1Excel\test.xlsx");
```

4. Excel で、**[ページレイアウト]**タブを選択し、**[枠線]**セクションにある**[印刷]**チェックボックスをオンにします。ワークシートは次のように表示されます。

	A	B	C	D	E
1		テスト1	テスト2		
2					
3				ページ1	
4				ページ2	
5					
6					
7					
8					

## ブック間での行のコピー (WPFのみ)

ブックの中の1シート内の行を別のブックにコピーするには、次の手順に従います。

1. C1.WPF.Excel.4.dll への参照を追加します。
2. 既存のブックをロードします。

```
C#
C1XLBook wb = new C1XLBook();
wb.Load(@"C:\C1Excel\test.xlsx");
```

3. **Test** という名前の **XLSheet** を含む新しいワークブックを作成します。

```
C#
C1XLBook xb = new C1XLBook();
xb.Sheets.Add("Test");
```

4. 既存のブックのシートから、新しいワークブックの **Test** シートに各行をコピーします。


```
C#
XLSheet source = wb.Sheets[0];
XLSheet dest = xb.Sheets["Test"];
for (int row = 0; row <= source.Rows.Count - 1; row++)
{
    for (int col = 0; col <= source.Columns.Count - 1; col++)
    {
        dest[row, col].Value = source[row, col].Value;
    }
}
```

5. 新しいワークブックを保存してから開きます。

```
C#
// ファイルを保存してから開きます
```

```
xb.Save(@"C:\C1Excel\test2.xlsx");  
System.Diagnostics.Process.Start(@"C:\C1Excel\test2.xlsx");
```

6. 新しいブックを開きます。最初のブックの行が新しいブックに追加されます。

 このトピックの内容は、ComponentOne for WPFにのみ適用されます。

## 小計の作成

次のコードは、ブックのセルを書式設定する例を示します。

1. [表示]→[コード]を選択し、次のステートメントをフォームの先頭に追加します。

### VisualBasic

```
Import C1.WPF.Excel
```

### C#

```
using C1.WPF.Excel;
```

2. C1.WPF.Excel.4.dll/C1.Silverlight.Excel.5.dll への参照を追加し、**C1XLBook** を作成します。

```
C#  
// 新しいワークブックを作成します  
C1XLBook book = new C1XLBook();
```

3. **Window\_Loaded** イベントに次のコードを追加します。

### VisualBasic

```
Private Sub Window_Loaded(sender As Object, e As EventArgs)  
    Dim book As New C1XLBook()  
    Dim sheet As XLSheet = book.Sheets(0)  
    Dim totalStyle As New XLStyle(book)  
    totalStyle.Font = New XLFont("Arial", 10, true, false)  
    sheet(2, 1).Value = "計算"  
    sheet(2, 2).Value = "ID"  
    sheet(3, 1).Value = 12  
    sheet(3, 2).Value = 17  
    sheet.Rows(3).OutlineLevel = 2  
    sheet.Rows(3).Visible = False  
    sheet(4, 1).Value = 12  
    sheet(4, 2).Value = 14  
    sheet.Rows(4).OutlineLevel = 2  
    sheet.Rows(4).Visible = False  
    sheet(5, 1).Value = "12 小計"  
    sheet(5, 1).Style = totalStyle
```



```

sheet(5, 2).Value = 31
sheet(5, 2).Formula = "SUBTOTAL(9,C4:C5)"
sheet.Rows(5).OutlineLevel = 1
sheet(6, 1).Value = 34
sheet(6, 2).Value = 109
sheet.Rows(6).OutlineLevel = 2
sheet(7, 1).Value = "34 小計"
sheet(7, 1).Style = totalStyle
sheet(7, 2).Value = 109
sheet(7, 2).Formula = "SUBTOTAL(9,C7:C7)"
sheet.Rows(7).OutlineLevel = 1
sheet(8, 1).Value = "合計"
sheet(8, 1).Style = totalStyle
sheet(8, 2).Value = 140
sheet(8, 2).Formula = "SUBTOTAL(9,C4:C7)"
sheet.Rows(8).OutlineLevel = 0
book.Save("C:\C1Excel\mybook.xlsx")
System.Diagnostics.Process.Start("C:\C1Excel\mybook.xlsx")
End Sub

```

## C#

```

private void Window_Loaded(object sender, EventArgs e)
{
    C1XLBook book = new C1XLBook();
    XLSheet sheet = book.Sheets[0];
    XLStyle totalStyle = new XLStyle(book);
    totalStyle.Font = new XLFont("Arial", 10, true, false);
    sheet[2, 1].Value = "計算";
    sheet[2, 2].Value = "ID";
    sheet[3, 1].Value = 12;
    sheet[3, 2].Value = 17;
    sheet.Rows[3].OutlineLevel = 2;
    sheet.Rows[3].Visible = false;
    sheet[4, 1].Value = 12;
    sheet[4, 2].Value = 14;
    sheet.Rows[4].OutlineLevel = 2;
    sheet.Rows[4].Visible = false;
    sheet[5, 1].Value = "12 小計";
    sheet[5, 1].Style = totalStyle;
    sheet[5, 2].Value = 31;
    sheet[5, 2].Formula = "SUBTOTAL(9,C4:C5)";
    sheet.Rows[5].OutlineLevel = 1;
    sheet[6, 1].Value = 34;
    sheet[6, 2].Value = 109;
    sheet.Rows[6].OutlineLevel = 2;
    sheet[7, 1].Value = "34 小計";
    sheet[7, 1].Style = totalStyle;
    sheet[7, 2].Value = 109;
    sheet[7, 2].Formula = "SUBTOTAL(9,C7:C7)";
    sheet.Rows[7].OutlineLevel = 1;
}

```

```
sheet[8, 1].Value = "合計";  
sheet[8, 1].Style = totalStyle;  
sheet[8, 2].Value = 140;  
sheet[8, 2].Formula = "SUBTOTAL(9,C4:C7)";  
sheet.Rows[8].OutlineLevel = 0;  
book.Save(@"C:\C1Excel\mybook.xlsx");  
System.Diagnostics.Process.Start(@"C:\C1Excel\mybook.xlsx");  
}
```

4. プログラムを実行します。スプレッドシートが開かれて、次のように表示されます。SUBTOTAL 関数は、指定された行の合計を取得します。

	A	B	C	D	E
1					
2					
3		計算	ID		
6		12 小計	31		
7		34	109		
8		34 小計	109		
9		合計	140		
10					
11					
12					
13					
14					

## セルの書式設定

ブックのセルを書式設定するには、次の手順に従います。

1. C1.WPF.Excel.4.dll/C1.Silverlight.Excel.5.dll への参照を追加し、**C1XLBook** を作成します。

```
C#  
  
// 保存する新しいワークブックを作成します  
C1XLBook book = new C1XLBook();
```

2. ワークブックにコンテンツを追加し、新しいスタイルを作成して、シートの最初の列のセルにそのスタイルを適用します。

```
C#  
  
// 新しいスタイルを作成します  
XLStyle style1 = new XLStyle(book);  
style1.ForeColor = Colors.Yellow;  
style1.BackColor = Colors.Blue;  
style1.Format = "$ .00";  
  
// コンテンツを追加し、シートの最初の列のセルにスタイルを適用します  
XLSheet sheet = book.Sheets[0];  
int i;  
  
for (i = 0; i <= 9; i++)  
{  
    sheet[i, 0].Value = i + 1;
```

```
sheet[i, 0].Style = style1;
}
```

3. ワークブックを保存してから開きます。

```
C#
// ファイルを保存してから開きます
book.Save(@"C:\C1Excel\test.xlsx");
System.Diagnostics.Process.Start(@"C:\C1Excel\test.xlsx");
```

	A	B
1	\$ 1.00	
2	\$ 2.00	
3	\$ 3.00	
4	\$ 4.00	
5	\$ 5.00	
6	\$ 6.00	
7	\$ 7.00	
8	\$ 8.00	
9	\$ 9.00	
10	\$ 10.00	
11		

## OpenXml ファイルのインポートとエクスポート (WPFのみ)

**Excel for WPF** では、Microsoft Excel 2007 OpenXml ファイルを読み書きできるようになりました。OpenXml は、Microsoft によって Office 2007 に導入されたオープンスタンダードベースの形式です。BIFF8 などの独自のバイナリ形式とは異なり、OpenXml は XML に基づき、ドキュメントも公開されているため、OpenXml ファイルはアプリケーションで簡単に操作できます。OpenXml ファイルには、Zip 圧縮されたいくつかの XML ファイルが含まれます。これらのファイルは圧縮されているため、通常、OpenXml ファイルは、.doc や .xls などの従来のドキュメントファイルより小さくなります。

**Excel for WPF** では、OpenXml ファイルのデータをロードおよび保存したり、情報を書式設定することはできますが、数式はロードまたは保存されません。BIFF 形式では数式が不透過としてコピーされますが、これは、現時点では OpenXML 形式でサポートされていません。数式を含むファイルをロードして保存すると、それらの数式は削除されます。この点は、数式を保持する従来の .xls または BIFF8 形式とは異なります。

OpenXml 形式をサポートするため、**C1XLBook** の **Load** メソッドと **Save** メソッドには、**FileFormat** パラメータを取るオーバーロードメソッドがあります。このパラメータは、ファイルのロードまたは保存時に、使用するファイル形式を指定するために使用されます。

ファイル名が指定されなかった場合、**Excel for WPF** は、ファイル名拡張子からファイル形式を推定します。デフォルトでは、"XLSX" および "ZIP" 拡張子を持つファイルは、OpenXml ファイルとしてロードおよび保存されます。その他のファイルは、BIFF8 または .xls 形式としてロードおよび保存されます。

次に例を示します。

```
C#
// ファイル拡張子に基づいてロードおよび保存します
book.Load("somefile.xls"); // biff 8 ファイルをロードします
book.Save("somefile.xlsx"); // ファイルを OpenXml として保存します
book.Save("somefile.zip"); // ファイルを OpenXml として保存します

// FileFormat を指定してロードおよび保存します
book.Load("somefile.xls", FileFormat.Biff8);
book.Save("somefile.xlsx", FileFormat.OpenXml);
```

ストリームからファイルをロードしたり、ストリームに保存するときに形式を指定することもできます。**FileFormat** が指定されなかった場合、**Excel for WPF** はデフォルトで BIFF8 形式を使用します。

ここには、小さな動作変更が含まれていることに注意してください。次のステートメントを考えます。

XAML

```
book.Save("somefile.xlsx");
```

これにより、これまでの Excel for WPF では、(間違った拡張子で)BIFF8 ファイルが保存されます。これからは、(正しい拡張子で)OpenXml ファイルが保存されます。アプリケーションにこのようなコードがある場合は、アップグレード時に次のように変更する必要があります。

XAML

```
// 意図的に間違った拡張子でファイルを保存します
book.Save("somefile.xlsx", FileFormat.Biff8);
```

**OpenXml ファイルにブックをエクスポートするには、次の手順に従います。**

1. 既存のブックをロードします。

## VisualBasic

```
Dim wb As New C1XLBook()
wb.Load("C:\C1Excel\test.xlsx")
' または
Dim wb As New C1XLBook()
wb.Load("C:\C1Excel\test.xlsx", C1.WPF.Excel.FileFormat.OpenXml)
```

## C#

```
C1XLBook wb = new C1XLBook();
wb.Load(@"C:\C1Excel\test.xlsx");
// または
C1XLBook wb = new C1XLBook();
wb.Load(@"C:\C1Excel\test.xlsx", C1.WPF.Excel.FileFormat.OpenXml);
```

2. ブックを OpenXml 形式ファイルにエクスポートします。


## VisualBasic

```
Dim wb As New C1XLBook()
' コンテンツを追加します
Dim sheet As XLSheet = wb.Sheets(0)
Dim i As Integer
For i = 0 To 9
    sheet(i,0).Value = i + 1
Next i
' OpenXml 形式ファイルにエクスポートします
```

```
wb.Save("C:\C1Excel\test.xlsx")
' または
' OpenXml 形式ファイルにエクスポートします
wb.Save("C:\C1Excel\test.xlsx", C1.WPF.Excel.FileFormat.OpenXml)
```

## C#

```
C1XLBook wb = new C1XLBook();
// コンテンツを追加します
XLSheet sheet = wb.Sheets[0];
for (int i = 0; i <= 9; i++)
{
    sheet[i,0].Value = i + 1;
}
// OpenXml 形式ファイルにエクスポートします
wb.Save(@"C:\C1Excel\test.xlsx");
// または
// OpenXml 形式ファイルにエクスポートします
wb.Save(@"C:\C1Excel\test.xlsx", C1.WPF.Excel.FileFormat.OpenXml);
```

 このトピックの内容は、ComponentOne for WPFにのみ適用されます。

## CSV ファイルの保存 (Silverlightのみ)

**Excel for Silverlight** は、カンマ区切り値 (CSV) ファイルの保存と読み込みをサポートします。CSV は、数値やテキストを含む表形式データを読みやすいプレーンテキスト形式で保存する一般的なファイル形式です。

次のコードは、.csv ファイルを保存する方法の例を示します。

1. **C1.Silverlight.Excel.5.dll** への参照を追加します。
2. 標準の **TextBox** コントロールをページに追加し、**txt\_Status** と名前を付けます。
3. **Button** コントロールをページに追加し、**\_btnSave** と名前を付けます。
4. **[表示]** → **[コード]** を選択し、次のステートメントをフォームの先頭に追加します。
  - Import C1.Silverlight.Excel(Visual Basic)
  - using C1.Silverlight.Excel;(iC#)
5. **\_book** という名前の新しい **C1XLBook** を作成し、いくつかの値を含むシートをそのブックに追加します。次のコードを使用します。

```
C#
public partial class MainPage : UserControl
{
    C1XLBook _book = new C1XLBook ();
    public MainPage ()
    {
        InitializeComponent ();

        XLSheet sheet = _book.Sheets[0];
```

```
        for (int i = 0; i <= 9; i++)
        {
            sheet[i, 0].Value = i + 1;
            sheet[i, 1].Value = 10 - i;
        }
    }
}
```

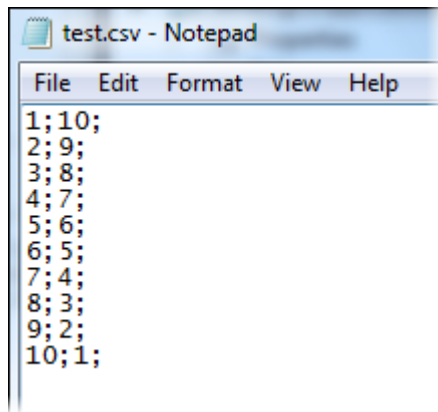
6. 次に、ユーザーがボタンをクリックしたときにブックを CSV ファイルに保存するコードを次のように追加します。ファイルの保存中および保存の終了時に、テキストボックスにその状態が表示されます。

## XAML

```
private void _btnSave_Click(object sender, RoutedEventArgs e)
{
    var dlg = new SaveFileDialog();
    dlg.Filter = "Comma-Separated Values (*.csv)|*.csv";
    if (dlg.ShowDialog().Value)
    {
        try
        {
            // 情報
            txt_Status.Text = string.Format("Saving {0}...",
            dlg.SafeFileName);

            // ワークブックを保存します
            using (var stream = dlg.OpenFile())
            {
                _book.Sheets[0].SaveCsv(stream);
            }
            txt_Status.Text = string.Format("Saved {0}",
            dlg.SafeFileName); ;
        }
        catch (Exception x)
        {
            MessageBox.Show(x.Message);
        }
    }
}
```

7. **[表示]**→**[デザイナー]**をクリックしてデザインビューに戻ります。
8. ボタンを選択し、Visual Studio のプロパティウィンドウで**[イベント]**タブをクリックします。
9. **[クリック]**イベントの横にあるドロップダウン矢印をクリックし、**\_btnSave\_Click** を選択して、追加したコードにイベントをリンクします。
10. **[F5]**キーを押してプロジェクトを実行し、ボタンをクリックします。
11. 任意の場所にファイルを保存してから、ファイルを開きます。次の画像のようになります。



 このトピックの内容は、ComponentOne for Silverlight にのみ適用されます。

## ワークブックの計算モードの設定

**CalculationMode** プロパティは、ワークブック内のすべての数式の計算モードを指定します。**CalculationMode** 列挙は、**Manual**(手作業で計算を実行する)、**Auto**(自動的に計算を実行する)、**AutoNoTable**(テーブル以外は計算を実行する)の3つのオプションを提供します。

計算モードを設定するには、次の手順に従います。

1. C1.WPF.Excel.4.dll/C1.Silverlight.Excel.5.dll への参照を追加し、**C1XLBook** を作成します。

```
C#
// 新しいワークブックを作成します
C1XLBook book = new C1XLBook();
```

2. 次のコードを使用して、簡単な数式を追加します。

```
C#
XLSheet sheet = book.Sheets[0];

// 簡単な数式
sheet[7, 0].Value = "数式: 5!";
sheet[7, 1].Value = 122;
sheet[7, 1].Formula = "1*2*3*4*5";
book.CalculationMode = CalculationMode.Auto;
```

3. .xlsx ファイルを保存してから開きます。

```
C#
// ファイルを保存してから開きます
book.Save(@"C:\C1Excel\test.xlsx");
System.Diagnostics.Process.Start(@"C:\C1Excel\test.xlsx");
```

**CalculationMode** を **Auto** に設定したため、セル(7,1)の値が **122** ではなく **120**( $1*2*3*4*5$  の結果)になっていることに注目してください。

# Excel for WPF/Silverlight

The image shows a screenshot of an Excel spreadsheet. The active cell is B8, which contains the text "数式: 5!". The formula bar at the top shows the formula "=1\*2\*3\*4\*5". The spreadsheet grid shows columns A through E and rows 1 through 11. Cell B8 is highlighted with a black border, and the value "120" is visible inside it.

	A	B	C	D	E
1					
2					
3					
4					
5					
6					
7					
8	数式: 5!	120			
9					
10					
11					