

入門指南

本指南將協助您在專案中設定 SpreadsheetExporter，並建立第一個 Excel 汇出功能。

安裝

安裝 NuGet 套件

1. 安裝核心函式庫：

```
dotnet add package CloudyWing.SpreadsheetExporter
```

2. 選擇並安裝 Excel 實作套件：

選項 1：NPOI (Apache POI for .NET)

```
dotnet add package CloudyWing.SpreadsheetExporter.Excel.NPOI  
dotnet add package NPOI
```

選項 2：EPPlus

```
dotnet add package CloudyWing.SpreadsheetExporter.Excel.EPPlus  
dotnet add package EPPlus
```

TIP

NPOI 和 EPPlus 皆受支援，請依據專案需求和授權考量選擇合適的套件。

設定

在應用程式啟動時（例如 `Program.cs`）加入以下設定：

```
using CloudyWing.SpreadsheetExporter;  
using CloudyWing.SpreadsheetExporter.Excel.NPOI;  
  
SpreadsheetManager.SetExporter(() => new ExcelExporter());
```

⊗ IMPORTANT

請務必使用 `SpreadsheetManager.CreateExporter()` 建立 Exporter 實例，而非直接使用 `new ExcelExporter()`。這樣可以輕鬆切換 Excel 實作套件，並套用全域設定。

第一個匯出範例

以下是建立 Excel 檔案的簡單範例：

```
// 建立 Exporter 實例
ExporterBase exporter = SpreadsheetManager.CreateExporter();

// 建立 Sheeter (代表一個工作表)
Sheeter sheeter = exporter.CreateSheeter();

// 建立 Template 並加入內容
GridTemplate template = new GridTemplate();
template.CreateRow();
template.CreateCell("Hello, SpreadsheetExporter!");

// 將 Template 加入 Sheeter
sheeter.AddTemplate(template);

// 匯出為檔案
exporter.ExportFile($"C:\Sample{exporter.FileNameExtension}");
```

下一步

現在您已經完成 SpreadsheetExporter 的設定，接下來可以深入了解核心概念：

- [Exporters](#) - 了解活頁簿層級的設定
- [Sheeters](#) - 理解工作表管理
- [Templates](#) - 探索不同的資料結構化方式
- [Customization](#) - 自訂儲存格樣式與格式

Exporters

ISpreadsheetExporter 介面代表一個活頁簿 (Workbook)，提供活頁簿層級的設定方法以及工作表管理功能。

設定

您可以透過 `SpreadsheetManager.SetExporter()` 全域設定 Exporter：

```
SpreadsheetManager.SetExporter(() => {
    ExcelExporter exporter = new ExcelExporter();
    exporter.Password = "Your Workbook Password";
    exporter.DefaultBasicSheetName = "工作表";

    return exporter;
});
```

可用屬性

| 屬性 | 說明 | --- | --- | **Password** | 設定活頁簿密碼保護 | **DefaultBasicSheetName** | 工作表名稱預設前綴（預設值：「工作表」）。自動產生的工作表名稱格式為 {前綴}{編號} | **ContentType** | 汇出格式的 MIME 類型（例如：application/vnd.openxmlformats-officedocument.spreadsheetml.sheet） | **FileNameExtension** | 汇出格式的副檔名（例如：.xlsx） |

事件

您可以透過事件掛勾匯出生命週期：

```
SpreadsheetManager.SetExporter(() => {
    ExcelExporter exporter = new ExcelExporter();

    exporter.SpreadsheetExportingEvent += (sender, e) => {
        // 在匯出開始前觸發
        Console.WriteLine("開始匯出...");
    };

    exporter.SpreadsheetExportedEvent += (sender, e) => {
        // 在匯出完成後觸發
        Console.WriteLine("匯出完成!");
    };
};

exporter.SheetCreatedEvent += (sender, e) => {
    // 在建立工作表時觸發
    // 可存取底層的工作表物件 (例如 NPOI 的 ISheet 或 EPPlus 的 ExcelWorksheet)
};
```

```
    var sheetObject = e.SheetObject;
    var context = e.SheetContext;
};

return exporter;
});
```

使用 Sheeters 建立 Sheeters

```
ISpreadsheetExporter exporter = SpreadsheetManager.CreateExporter();

// 建立指定名稱的工作表
Sheeter sheeter = exporter.CreateSheeter("測試工作表");

// 建立自動命名的工作表 (例如「工作表1」)
Sheeter sheeter2 = exporter.CreateSheeter();
```

取得 Sheeters

```
// 取得最後一個 Sheeter (若無則建立一個)
Sheeter lastSheeter = exporter.LastSheeter;

// 依索引取得 Sheeter
Sheeter firstSheeter = exporter.GetSheeter(0);
```

匯出

匯出至位元組陣列

當不需要儲存到磁碟時 (例如網頁下載) , 使用 `Export()`:

```
public IActionResult Download() {
    ISpreadsheetExporter exporter = SpreadsheetManager.CreateExporter();
    exporter.CreateSheeter();

    return File(
        exporter.Export(),
        exporter.ContentType,
        $"Spreadsheet{exporter.FileNameExtension}"
    );
}
```

匯出至檔案

使用 `ExportFile()` 儲存到磁碟：

```
ISpreadsheetExporter exporter = SpreadsheetManager.CreateExporter();
exporter.CreateSheeter();

// 若檔案存在則覆寫
exporter.ExportFile($"C:\Sample{exporter.FileNameExtension}");

// 若檔案存在則拋出例外
exporter.ExportFile(
    $"C:\Sample{exporter.FileNameExtension}",
    Spreadsheet FileMode.CreateNew
);
```

Spreadsheet FileMode 選項：

- `Create` - 建立新檔案，若存在則覆寫（預設）
- `CreateNew` - 建立新檔案，若存在則拋出例外

相關主題

- [入門指南](#) - 從零開始設定 `SpreadsheetExporter`
- [Sheeters](#) - 學習如何為 `Exporter` 建立與管理工作表
- [Templates](#) - 了解如何在 `Sheeter` 中使用各種範本類型
- [自訂樣式](#) - 透過 `SpreadsheetManager` 設定全域儲存格樣式

Sheeters

Sheeter 代表一個工作表，包含產生工作表內容所需的設定與 Templates。

基本設定

變更工作表名稱

建立 Sheeter 後仍可修改工作表名稱：

```
Sheeter sheeter = exporter.CreateSheeter();
sheeter.SheetName = "新工作表1";
```

設定工作表密碼

為工作表設定密碼保護：

```
Sheeter sheeter = exporter.CreateSheeter();
sheeter.Password = "Your Sheet Password";
```

您可以透過擴充方法簡化此模式：

```
public static class ExporterBaseExtensions {
    public static Sheeter CreateSheeterWithPassword(
        this ExporterBase exporter,
        string sheetName = null
    ) {
        Sheeter sheeter = exporter.CreateSheeter(sheetName);
        sheeter.Password = "Your Sheet Password";
        return sheeter;
    }
}
```

欄位設定

設定欄寬

使用 `SetColumnWidth()` 設定欄寬（欄位索引從 0 開始）：

```
Sheeter sheeter = exporter.CreateSheeter();

// 設定特定寬度
sheeter.SetColumnWidth(0, 10d);
```

```
// 隱藏欄位  
sheeter.SetColumnWidth(1, Constants.HiddenColumn);  
  
// 自動調整欄寬  
sheeter.SetColumnWidth(2, Constants.AutoFitColumnWidth);
```

使用 Templates 加入 Templates

Templates 會垂直堆疊。若 `template1` 佔用 3 列，`template2` 將從第 4 列開始。

```
sheeter.AddTemplate(template1);  
sheeter.AddTemplate(template2);  
  
// 一次加入多個 Templates  
sheeter.AddTemplates(template3, template4);
```

NOTE

Templates 定義工作表的結構與內容。請參閱 [Templates 指南](#) 了解 more 資訊。

相關主題

- [入門指南](#) - 了解 Sheeter 在整體匯出流程中的角色
- [Exporters](#) - 學習如何透過 Exporter 建立 Sheeter
- [Templates](#) - 探索可加入 Sheeter 的各種範本類型
- [自訂樣式](#) - 設定 Sheeter 中儲存格的預設樣式

Templates

Templates 定義工作表儲存格的結構與內容。SpreadsheetExporter 提供多種 Template 類型以滿足不同使用情境。

本文內容

- [GridTemplate](#) - 手動逐格配置
- [DataTableTemplate](#) - 基於 DataTable 的匯出
- [RecordSetTemplate](#) - 強型別集合
- [MergedTemplate](#) - 合併多個 Templates
- [自訂 Templates](#) - 建立您自己的 Template

GridTemplate

[GridTemplate](#) 提供精細的儲存格配置控制，類似於 HTML 的 `<table>`、`<tr>` 與 `<td>`。支援方法鏈 (Method Chaining) 以簡化程式碼。

建立列

```
GridTemplate template = new GridTemplate();

// 預設列高
template.CreateRow();

// 指定列高
template.CreateRow(20d);

// 隱藏列
template.CreateRow(Constants.HiddenRow);

// 自動調整列高
template.CreateRow(Constants.AutoFitRowHeight);
```

建立儲存格

```
GridTemplate template = new GridTemplate();
template.CreateRow();

// 簡單儲存格
template.CreateCell("Value1_1");

// 合併儲存格 (RowSpan=3, ColumnSpan=2)
template.CreateCell("Value1_2", 3, 2);
```

```

// 自訂樣式的儲存格
template.CreateCell("Value1_3", 1, 1, new CellStyle());

// 使用方法鏈與公式
template.CreateRow()
    .CreateCell("Value2_1")
    .CreateCell((cell, row) => $"{cell} + {row}"); // 公式 (索引從 0 開始)

/*
輸出:
+-----+-----+-----+-----+-----+
| Value1_1 | Value1_2 |       | Value1_3 |
+-----+           |-----+       |
| Value2_1 |           | =1 + 1 |
+-----+-----+-----+-----+-----+
*/

```

DataTableTemplate

直接將 `System.Data.DataTable` 汇出至 Excel，並自動對應欄位。

```

// 建立 DataTable
System.Data.DataTable dataTable = new System.Data.DataTable();
dataTable.Columns.Add("Name", typeof(string));
dataTable.Columns.Add("Age", typeof(int));

dataTable.Rows.Add("John", 30);
dataTable.Rows.Add("Mary", 25);

// 建立 Template
DataTableTemplate template = new DataTableTemplate(dataTable);

// 設定列高
template.HeaderHeight = 25;
template.RecordHeight = 20;

/*
輸出:
+---+---+
| Name | Age |
+---+---+
| John | 30 |
+---+---+
| Mary | 25 |
+---+---+

```

```
| ---- | --- |  
*/
```

RecordSetTemplate

最強大的 Template，適用於強型別資料集合。提供完整的欄位設定、資料轉換與樣式控制。

基本用法

```
public class Record {  
    public int Id { get; set; }  
    public string Name { get; set; }  
}  
  
List<Record> source = new List<Record> {  
    new Record { Id = 0, Name = "Marry" },  
    new Record { Id = 1, Name = "Terry" }  
};  
  
RecordSetTemplate<Record> template = new RecordSetTemplate<Record>(source);  
template.Columns.Add("編號", x => x.Id);  
template.Columns.Add("姓名", x => x.Name);  
  
/*  
輸出:  
| ---- | ---- |  
| 編號 | 姓名 |  
| ---- | ---- |  
| 0     | Marry |  
| ---- | ---- |  
| 1     | Terry |  
| ---- | ---- |  
*/
```

資料轉換

在匯出過程中轉換值：

```
// 轉換屬性值  
template.Columns.Add("大寫姓名", x => x.Name, x => x.UseValue(y => y.Value.ToUpper()));  
  
// 合併多個屬性  
template.Columns.Add("合併資料", x => x.UseValue(y => y.Record.Id + y.Record.Name));  
  
/*
```

輸出:

大寫姓名	合併資料
MARRY	0Marry
TERRY	1Terry

*/

自訂儲存格樣式

套用條件樣式:

```
CellStyleConfiguration cellStyles = SpreadsheetManager.DefaultCellStyles;

template.Columns.Add(
    "狀態",
    x => x.Id,
    // 標題樣式: 紅色背景
    cellStyles.HeaderStyle with {
        BackgroundColor = Color.Red
    },
    // 條件式欄位樣式
    x => x.Value == 0
        ? cellStyles.FieldStyle with { BackgroundColor = Color.Blue }
        : cellStyles.FieldStyle with { BackgroundColor = Color.Yellow }
);
```

多層標題

建立分組欄位標題:

```
template.Columns.Add("群組1")
    .AddChildToLast("編號", x => x.Id)
    .AddChildToLast("姓名", x => x.Name)
    .Add("群組2")
    .AddChildToLast("子群組1");

DataColumnCollection<Record> lastChildColumns = template.Columns.Last().ChildColumns;
lastChildColumns.AddChildToLast("編號", x => x.Id)
    .AddChildToLast("姓名", x => x.Name);

template.Columns.AddChildToLast("子群組2");
lastChildColumns = template.Columns.Last().ChildColumns
```

```

.AddChildToLast("編號", x => x.Id)
.AddChildToLast("姓名", x => x.Name);

/*
輸出:
+---+---+---+---+---+---+
|       |       群組2      | |
| 群組1 |-----|-----|
|       | 子群組1 | 子群組2 |
+---+---+---+---+---+
| 編號 | 姓名 | 編號 | 姓名 | 編號 | 姓名 |
+---+---+---+---+---+---+
*/

```

公式

在儲存格中使用 Excel 公式：

```

template.Columns.Add("公式", x => x.UseFormula(y => $"{y.CellIndex} + {y.RowIndex}"));

/*
輸出 (第一筆資料在索引 1) :
+---+
| 公式 |
+---+
| =0+1 |
+---+
| =0+2 |
+---+
*/

```

凍結窗格與自動篩選

```

RecordSetTemplate<Record> template = new RecordSetTemplate<Record>(source);
// ... 加入欄位 ...

// 凍結標題列 (依據標題層數自動決定)
template.IsFreezeHeader = true;

// 啟用自動篩選 (包含標題與資料範圍)
template.IsAutoFilterEnabled = true;

// 設定列高

```

```
template.HeaderHeight = 30;  
template.RecordHeight = 25;
```

資料驗證

為儲存格加入資料驗證規則以限制輸入。

在 RecordSetTemplate 中使用

```
template.Columns.Add("年齡", x => x.Age, provider => provider.UseDataValidation(x => new  
DataValidation {  
    ValidationType = DataValidationType.Integer,  
    Operator = DataValidationOperator.Between,  
    Value1 = 18,  
    Value2 = 65,  
    ErrorTitle = "輸入錯誤",  
    ErrorMessage = "年齡必須在 18 到 65 歲之間",  
    IsErrorAlertShown = true  
}));
```

在 GridTemplate 中使用

```
GridTemplate template = new GridTemplate();  
template.CreateRow();  
template.CreateCell(cell => {  
    cell.ValueGenerator = (c, r) => "請選擇";  
    cell.DataValidationGenerator = (c, r) => new DataValidation {  
        ValidationType = DataValidationType.List,  
        ListItems = new[] { "選項 A", "選項 B", "選項 C" },  
        IsDropdownShown = true  
    };  
});
```

MergedTemplate

合併多個 Templates。適用於建立複雜版面配置。

```
MergedTemplate merged = new MergedTemplate();  
merged.AddTemplate(headerTemplate);  
merged.AddTemplate(dataTemplate);  
merged.AddTemplate(footerTemplate);  
  
sheeter.AddTemplate(merged);
```

自訂 Templates

實作 `ITemplate` 介面以建立可重複使用的自訂 Template：

```
public class ReportInfoTemplate : ITemplate {
    private readonly GridTemplate gridTemplate = new GridTemplate();

    public ReportInfoTemplate(string title, string user, int colSpan) {
        CellStyleConfiguration cellStyles = SpreadsheetManager.DefaultCellStyles;
        CellStyle titleStyle = cellStyles.CellStyle with {
            HorizontalAlignment = HorizontalAlignment.Center,
            Font = cellStyles.CellStyle.Font with { Size = 14 }
        };

        // 標題列
        gridTemplate.CreateRow();
        gridTemplate.CreateCell(title, colSpan, cellStyle: titleStyle);

        // 資訊列
        int leftColSpan = colSpan / 2;
        gridTemplate.CreateRow();
        gridTemplate.CreateCell($"使用者: {user}", leftColSpan,
cellStyle: cellStyles.CellStyle);
        gridTemplate.CreateCell(
            $"產生時間: {DateTime.Now:yyyy-MM-dd HH:mm:ss}",
            colSpan - leftColSpan,
            cellStyle: cellStyles.CellStyle
        );
    }

    public TemplateContext GetContext() {
        return gridTemplate.GetContext();
    }
}
```

相關主題

- [入門指南](#) - 了解如何在專案中開始使用 Templates
- [Exporters](#) - 學習 Template 與 Exporter 的整體協作流程
- [Sheeters](#) - 了解如何將 Templates 加入 Sheeter
- [自訂樣式](#) - 為 Template 中的儲存格套用自訂樣式

自訂樣式

專案使用的儲存格樣式皆定義在 `SpreadsheetManager.DefaultCellStyles`，提供預設樣式給不同元件使用。

預設樣式

`CellStyleConfiguration` 提供以下預設樣式：

樣式	用途
<code>CellStyle</code>	預設儲存格樣式
<code>GridCellStyle</code>	<code>GridTemplate</code> 使用的預設樣式
<code>HeaderStyle</code>	<code>RecordSetTemplate</code> 標題列使用的預設樣式
<code>FieldStyle</code>	<code>RecordSetTemplate</code> 資料列使用的預設樣式

更改預設樣式

基於現有樣式修改

以下範例將所有預設樣式的字體大小放大：

```
SpreadsheetManager.DefaultCellStyles = new CellStyleConfiguration((setuper) => {
    CellStyleConfiguration oldCellStyles = SpreadsheetManager.DefaultCellStyles;

    setuper.CellStyle = oldCellStyles.CellStyle with {
        Font = oldCellStyles.CellStyle.Font with { Size = 16 }
    };

    setuper.GridCellStyle = oldCellStyles.GridCellStyle with {
        Font = oldCellStyles.GridCellStyle.Font with { Size = 14 }
    };

    setuper.HeaderStyle = oldCellStyles.HeaderStyle with {
        Font = oldCellStyles.HeaderStyle.Font with { Size = 14 }
    };

    setuper.FieldStyle = oldCellStyles.FieldStyle with {
        Font = oldCellStyles.FieldStyle.Font with { Size = 14 }
    };
});
```

完整自訂

您也可以使用 `new CellStyle()` 完整自訂所有屬性：

```
SpreadsheetManager.DefaultCellStyles = new CellStyleConfiguration((setuper) => {
    CellFont customFont = new CellFont(
        fontName: "微軟正黑體",
        size: 12,
        color: Color.Black,
        style: FontStyles.None
    );

    setuper.CellStyle = new CellStyle(
        horizontalAlignment: HorizontalAlignment.Left,
        verticalAlignment: VerticalAlignment.Middle,
        hasBorder: false,
        wrapText: false,
        backgroundColor: Color.Empty,
        font: customFont
    );

    // ... 設定其他樣式 ...
});
```

從設定檔讀取樣式

設定檔結構

在專案中建立 `Spreadsheet.json` 並設定「複製到輸出目錄」：

```
{
    "Spreadsheet": {
        "Cell": {
            "HorizontalAlignment": "Center",
            "VerticalAlignment": "Middle",
            "HasBorder": false,
            "WrapText": false,
            "Font": {
                "FontName": "新細明體",
                "FontSize": 10,
                "IsBold": false,
                "IsItalic": false,
                "HasUnderline": false,
                "IsStrikeout": false
            }
        }
    }
}
```

```
        }
    }
}
```

載入設定

在 Program.cs 中讀取並套用設定：

```
public class ConfigSettings {
    public CellSettings Cell { get; set; }
}

public class CellSettings {
    public HorizontalAlignment HorizontalAlignment { get; set; }
    public VerticalAlignment VerticalAlignment { get; set; }
    public bool HasBorder { get; set; }
    public bool WrapText { get; set; }
    public FontSettings Font { get; set; }
}

public class FontSettings {
    public string FontName { get; set; }
    public short FontSize { get; set; }
    public bool IsBold { get; set; }
    public bool IsItalic { get; set; }
    public bool HasUnderline { get; set; }
    public bool IsStrikeout { get; set; }
}

// 建立設定載入器
IConfigurationRoot config = new ConfigurationBuilder()
    .SetBasePath(AppDomain.CurrentDomain.BaseDirectory)
    .AddJsonFile("Spreadsheet.json", optional: false, reloadOnChange: true)
    .Build();

// 設定自動重新載入
ChangeToken.OnChange(() => config.GetReloadToken(), () => {
    Initialize();
});

Initialize();

void Initialize() {
    ConfigSettings configSettings = config.GetSection("Spreadsheet").Get<ConfigSettings>();
    CellSettings cellSettings = configSettings.Cell;
```

```

// 組合字型樣式
FontStyles fontStyle = FontStyles.None;
if (cellSettings.Font.IsBold) {
    fontStyle |= FontStyles.IsBold;
}
if (cellSettings.Font.IsItalic) {
    fontStyle |= FontStyles.IsItalic;
}
if (cellSettings.Font.HasUnderline) {
    fontStyle |= FontStyles.HasUnderline;
}
if (cellSettings.Font.IsStrikeout) {
    fontStyle |= FontStyles.IsStrikeout;
}

// 建立儲存格樣式
CellStyle cellStyle = new CellStyle(
    cellSettings.HorizontalAlignment,
    cellSettings.VerticalAlignment,
    cellSettings.HasBorder,
    cellSettings.WrapText,
    Color.Empty,
    new CellFont(
        cellSettings.Font.FontName,
        cellSettings.Font.FontSize,
        Color.Black,
        fontStyle
    )
);
;

// 套用至全域設定
SpreadsheetManager.DefaultCellStyles = new CellStyleConfiguration((setuper) => {
    setuper.CellStyle = cellStyle;
    setuper.GridCellStyle = cellStyle;
    setuper.HeaderStyle = cellStyle with {
        Font = cellStyle.Font with {
            Style = cellStyle.Font.Style | FontStyles.IsBold
        },
        HorizontalAlignment = HorizontalAlignment.Center,
        HasBorder = true
    };
    setuper.FieldStyle = cellStyle with {
        HorizontalAlignment = HorizontalAlignment.General,
        HasBorder = true
    };
});

```

```
});  
}
```

 **TIP**

使用 `ChangeToken.OnChange` 可在設定檔變更時自動重新載入樣式，無需重新啟動應用程式。

相關主題

- [入門指南](#) - 了解 SpreadsheetManager 的設定時機
- [Exporters](#) - 學習如何在 Exporter 層級套用全域樣式
- [Sheeters](#) - 了解 Sheeter 如何使用預設樣式
- [Templates](#) - 學習在 Template 中覆寫或使用自訂樣式