

Table of Contents

前言	1.1
一、基本概念	1.2
1. UDE-Report	1.2.1
範例：AbstractReport	1.2.1.1
2. 環境設定與測試	1.2.2
環境設定	1.2.2.1
單元測試	1.2.2.2
關於說明文件的其它範例	1.2.2.3
3. 文件格式(PDF/Word/RTF)	1.2.3
4. 試算表格式(Excel/ODS)	1.2.4
5. CSV格式	1.2.5
二、頁面控制	1.3
1. 內容分節	1.3.1
2. 邊界與換頁	1.3.2
3. 頁首頁尾	1.3.3
基本文字格式	1.3.3.1
基本頁次資訊	1.3.3.2
PDF支援 # TODO	1.3.3.3
4. 浮水印 # TODO	1.3.4
三、文字	1.4
範例：字型與字面	1.4.1
範例：ParagraphBuilder	1.4.2
補充：字型管理實作	1.4.3
四、表格	1.5
1. 基本表格輸出	1.5.1
範例：建立基本 PDF 表格	1.5.1.1
範例：Excel 表格欄位輸出	1.5.1.2
2. 表格欄位樣式	1.5.2
範例：底色	1.5.2.1
範例：邊框	1.5.2.2
範例：對齊	1.5.2.3

3. 表格合併欄位	1.5.3
範例：PDF 跨欄	1.5.3.1
範例：PDF 子表格	1.5.3.2
範例：Excel 合併儲存格	1.5.3.3
4. 欄位資料來源	1.5.4
5. 特殊樣式處理	1.5.5
範例：切割欄位(Cell)	1.5.5.1
範例：線段外觀(Cell)	1.5.5.2
範例：外框處理(Table)	1.5.5.3
範例：每列底色變化(Table)	1.5.5.4
五、繪圖、定位輸出	1.6
1. 線段輸出	1.6.1
2. 文字輸出	1.6.2
3. 圖片輸出	1.6.3
範例：Barcodes	1.6.3.1
4. 表格輸出	1.6.4
六、表格描述定義	1.7
1. 資料表格	1.7.1
樹狀表格	1.7.1.1
範例：多層標題	1.7.1.1.1
範例：欄寬	1.7.1.1.2
範例：其它欄位插入模式	1.7.1.1.3
巢狀表格	1.7.1.2
資料來源	1.7.1.3
2. 額外區塊	1.7.2
3. 資料統計	1.7.3
4. 其它轉換器	1.7.4
七、其它議題	1.8
1. PDF文件安全性設定	1.8.1
2. PDFPageEvent	1.8.2
3. 大型資料表格	1.8.3

UDE - Reports 說明文件與範例

目前相關 **libraries** 僅發佈於本公司內網，故所有相關範例應於內網環境執行。

- 範例程式 @ **GitHub** : <https://github.com/tcf625/ude-reports/>
- **Maven Repository Setting** :

```
<repositories>
  <repository>
    <id>IISI_UDE</id>
    <url>http://192.168.57.21/nexus/content/groups/public</url>
    <releases>
      <enabled>true</enabled>
    </releases>
    <snapshots>
      <enabled>true</enabled>
    </snapshots>
  </repository>
</repositories>
```

概述

UDE-Reports 報表產製輔助套件基於 **iText** 及 **Apache POI** 開發，主要目的為簡化使用原生 **API** 產製 **PDF** 及 **Excel** 文件的程式語法。另外，針對基本的版面 **Layout** 及 表格，可使用同一組描述資料與資料集，以產出不同格式的輸出結果，減少重複程式開發。

特性

- **PDF** 文件產製、條碼、浮水印及檔案上鎖等功能。
- 通用性表格定義描述 (轉 **PDF/EXCEL/CSV** 表格)。
- 通用表格裝飾元件。
- 多樣化 **PDF** 頁首頁尾支援。
- 可精準控制 **PDF** 圖形、文字輸出。
- 支援 **unicode** 輸出非0字面中文 (2字面、15字面)。
- 保留使用原生 **API** 彈性。
- 提供國內專案常見表樣的支援性。現成的報表軟體，也許只要用到一小部分功能，便能達成一般專案報表的多數需求；但本元件是處理其它較複雜需求所累積的經驗成果。

系統需求

- **JDK 8**

- iText 2.1.7
- POI
- 全字庫中文字型檔(選配)

基本概念

以下說明 **UDE-REPORT** 的文件結構及目前支援輸出格式的基本概念。

UDE Reports

文件結構

不同的文件格式，因為用途不同，所以支援的特性與文件結構多少有所差異。例如 **PDF/WORD** 文件用於輸出已排版完成的文件；**CSV** 格式通常作為資料交換用途，提供其它程式讀取；**EXCEL/ODS** 試算表格式兼具編輯與輸出用途，若以 **CSV** 格式代替不盡然可以滿足使用者需求。

在過去經歷專案中，許多報表軟體號稱可以用同一份表樣輸出為不同格式的文件，但通常只是形似而神不似。例如一張較複雜的 **PDF** 報表輸出為 **EXCEL** 時，可能會用多個儲存格合併輸出單一欄位值或者以重複輸出 **PAGE-HEADER / COLUMN-HEADER** 儲存格。這樣雖然印表輸出的結果看起來一樣，但是使用者卻難以用於後續的編修處理。最後為了確實滿足客戶需求，製表人員還是得針對不同格式，分別產生不同的表樣。

範例：隨意拉出的報表樣式，產為EXCEL格式時，出現過多的合併儲存格

	A	B	C	D	E	F	G
1							
2		PAGE-HEADER					

針對過去種種專案需求，**UDE-Report** 方案是使用底層 **API**—如 **itext**、**poi**，產出目的文件，並透過簡化**API**、規劃類別結構等等方法，減少處理文件一致性所付出的開發成本。

表樣

UDE-Report 相較其它市面上報表軟體的主要差異，在於以實作 **java** 程式類別設計表樣，而非提供視覺化環境編輯表樣 **metadata** (**xml** 或其它自定格式)。因為過往專案中，總是有部分需求必須以客製化程式處理，**UDE-Report**為因應這些客製化需求，其設計重點便在於簡化及統整第三方元件的 **API** 使用模式。

但使用**java**程式設計表樣，同樣會遇到屬於 **Java** 的程式議題，不良的 **coding** 成品容易造成表樣微調困難，開發／維護門檻較其它視覺性工具為高。團隊必須維持一份良好的設計原則、編程慣例，以保障相關表樣程式的可維護性。

Java 類別結構

文件內容輸出：**xxxDocument**

UDE-Report 針對支援的文件格式，提供對應的 **Document** 類別，如**PDFDocument**、**ExcelDocument**，只要建立類別實例，即可輕易使用各自提供的函式，輸出對應文件檔案內容。**CSV** 格式則因為操作方式單純，故直接使用 **Apache Commons** 中的 **CSVPrinter** 輸出。

文件檔案輸出：**xxxGenerator**

更進一步包裝文件產出類別，便是 **DocumentGenerator** —— 文件產生器的最上層介面。這個架構處理檔案 IO 開檔到關檔之間的雜務，還有多個文件分節組裝的相關機制。依目前對文件格式的支援，分別有 **PDFGenerator**、**ExcelGenerator**、**CSVGenerator** 等介面，自訂的表樣類別只要實作以下函式，產出文件內容即可。

```
void generatePDFContent(PDFDocument pdfDocument);           // PDFGenerator
void generateExcelContent(ExcelDocument<?, ?> document);     // ExcelGenerator
or
void generateCSVContent(CSVPrinter csvPrinter) throws IOException; // CSVGenerator
```

典型的專案會據以設計一或多個抽象類別，以處理專案自有的業務邏輯。

UDE-Report 是對於第三方製表元件的通用包裝，至於專案各自的一致性需求，就需要 **SA/SD** 的規劃。

一般來說，會有一個共用的最上層抽象報表類別，實作所有格式輸出介面，以負責專案中的共通需求：可能是一些 **protected** 屬性或函式。其下可能會再依文件性質，如：統計表、申請書(收據/收執聯)、資料清冊、套表輸出...，再拆分繼承架構，以統一頁首、頁尾、浮水印等設計樣式。也會有一個或多個清單 **Enum** 類別，定義專案內所有表樣，並載明共通特性定義，如報表名稱、輸出結果保存方式、浮水印樣式等，使共通的抽象表樣類別據以完成輸出。當然，還可以規劃報表輸入參數類別架構，用於傳入那些執行期才可以決定的參數內容。

以下是一個簡單的參考範例，實際使用時，請務必依實際需求重新規劃。

- 共通文件結構範例：[AbstractReport](#)

文件架構範例

所有範例原始碼，皆放於 github 專案：<https://github.com/tcf625/ude-reports/> 以供下載。

- 範例程式：`/ude-report-sample`
- 相關產出結果：`/sample-output`

表樣清單介面及定義範例

介面：**ReportDefinition**

基本的規劃範例：每一張表樣應有的資訊包括「代碼」、「名稱」及「可支援輸出格式」。

```
public interface ReportDefinition {  
    /** 代碼. */  
    String getReportCode();  
    /** 名稱. */  
    String getReportName();  
    /** 支援輸出格式. */  
    Set<DocumentFormat> getSupportedFormats();  
    /** 也可以統一處理輸出時使用的檔名(舉例). */  
    default String toFileName(final DocumentFormat format) {  
        final LocalDateTime localDateTime = Now.localDateTime();  
        final String time = RocDateUtils.format(localDateTime, "yyyMMdd-h-m-s");  
        return getReportCode() + "_" + time + "_" + uuid() + "." + format.getExtFi  
leName();  
    }  
}
```

定義：**AllReports**

這個範例只定義兩張表樣。一般正式的專案裡，表樣數量可能是數以百計，也有可能依業務性質拆分為多個不同的 **ENUM** 定義。

```
public enum AllReports implements ReportDefinition {  
    GSS0010("折抵役期空白清冊"), //  
    GSS0011("折抵役期資料匯出", DocumentFormat.CSV), //  
    ;  
  
    private final String reportName;  
    private final Set<DocumentFormat> supportedFormats;  
  
    /** 規劃預期支援的文件格式為 PDF/EXCEL. */  
    private AllReports(String reportName) {  
        this.reportName = reportName;  
    }  
}
```

```

        this.supportedFormats = EnumSet.of(DocumentFormat.PDF, DocumentFormat.EXCEL);
    }

    /** 自訂支援文件格式。 */
    private AllReports(String reportName, DocumentFormat first, DocumentFormat...
rest) {
        this.reportName = reportName;
        this.supportedFormats = EnumSet.of(first, rest);
    }

    @Override
    public String getReportName() {
        return this.reportName;
    }

    @Override
    public String getReportCode() {
        return name();
    }

    @Override
    public Set<DocumentFormat> getSupportedFormats() {
        return this.supportedFormats;
    }
}

```

文件產出範例

共用文件父類別：**AbstractSampleReport**

要求子類別實作所有輸出格式介面。實務上，reportDefinition、pageSize 也可用 Introduce Parameter Object 手法整合為同一參數物件。

```

public abstract class AbstractSampleReport extends AbstractPDFGenerator implements
ExcelGenerator, CSVGenerator {

    protected final SampleReportDefinition reportDefinition;

    protected AbstractSampleReport(SampleReportDefinition reportDefinition, Rectan
gle pageSize) {
        this.reportDefinition = reportDefinition;
        super.setPageSize(pageSize);
    }

    public SampleReportDefinition getReportDefinition() {

```

```

        return this.reportDefinition;
    }

    // ... 後略
}

```

GSS0010 / GSS0011

隨意輸出簡單內容，如：

```

@Override
public void generatePDFContent(final PDFDocument pdfDocument) {
    pdfDocument.writeText("TEST-GSS0010");
}

@Override
public void generateExcelContent(final ExcelDocument<?, ?> document) {
    final ExcelSheet<?> sheet = super.createExcelSheet(document);
    sheet.appendCell(new ExcelPoint(0, 0), "TEST-GSS0010", new CellFormat(Border.BOX));
    sheet.setColumnWidth(0, 20);
}

@Override
public void generateCSVContent(final CSVPrinter csvPrinter) throws IOException
{
    csvPrinter.print("TEST-GSS0010");
    csvPrinter.print("TEST-GSS0011");
    csvPrinter.print("TEST-GSS0012");
    csvPrinter.println();
}

```

GSS0010Test / GSS0011Test

用於執行報表程式，完整說明請參考後續「單元測試」範例。

環境與設定

Ude-Report 基於 UDE 套件開發，一般運作於 Spring 環境之下，但需準備的元件只有 PDFDocumentManager 一個。

```
<bean class="com.iisigroup.ude.report.itext2.PDFDocumentManager" />
```

預設使用隨同發布的 'classpath:itext-config-default.properties' 做為設定檔，其完整內容於下節說明。但其中有幾項定義需用到宣告 UdeBasicConfiguration 後建立的路徑環境變數 `${resource.path}` 及 `${global.resource.path}`。若希望脫離 UDE 甚或 Spring 環境執行，就必須自行修改設定檔，並用以建立 PDFDocumentManager 元件。

```
<bean class="com.iisigroup.ude.report.itext2.PDFDocumentManager">
  <constructor-arg index="0" value="classpath:itext-config-sris3.properties" />
</bean>
```

設定檔

設定檔使用 UDE-Extended properties 格式撰寫，預設的 itext-config-default.properties 內容如下：

```
# 預設字型
default.font=WindowsFont.MINGLIU
default.font.size=12

# 預設邊界
default.document.marginLeft=30
default.document.marginRight=30
default.document.marginBottom=30
default.document.marginTop=30

default.encryption.enable=false
default.encryption.owner.password=
default.encryption.user.password=

watermarks.config.path=${resource.path}/reports/marks/

#字型定義
font.WindowsFont.MINGLIU.0 = C:/windows/fonts/mingliu.ttc
font.WindowsFont.MINGLIU.0 = D:/windows/fonts/mingliu.ttc
font.WindowsFont.MINGLIU.0 = C:/winnt/fonts/mingliu.ttc
font.WindowsFont.MINGLIU.0 = D:/winnt/fonts/mingliu.ttc
font.WindowsFont.MINGLIU.0 = ${global.share.path}/fonts/mingliu.ttc

font.WindowsFont.MINGLIU.2 = C:/windows/fonts/mingliub.ttc
font.WindowsFont.MINGLIU.2 = D:/windows/fonts/mingliub.ttc
font.WindowsFont.MINGLIU.2 = C:/winnt/fonts/mingliub.ttc
font.WindowsFont.MINGLIU.2 = D:/winnt/fonts/mingliub.ttc
font.WindowsFont.MINGLIU.2 = ${global.share.path}/fonts/mingliub.ttc

font.WindowsFont.KAI.0 = C:/windows/fonts/kaiu.ttf
font.WindowsFont.KAI.0 = D:/windows/fonts/kaiu.ttf
font.WindowsFont.KAI.0 = C:/winnt/fonts/kaiu.ttf
font.WindowsFont.KAI.0 = D:/winnt/fonts/kaiu.ttf
font.WindowsFont.KAI.0 = ${global.share.path}/fonts/kaiu.ttf
#
# 依 http://www.cns11643.gov.tw/AIDB/cns\_authorization\_apply.do
# 全字庫字型逕行至「政府資料開放平臺」http://data.gov.tw/node/5961下載
#
font.CNS11643.SUNG.0=${global.resource.path}/fonts/TW-Sung-98_1.ttf
font.CNS11643.SUNG.2=${global.resource.path}/fonts/TW-Sung-Ext-B-98_1.ttf
font.CNS11643.SUNG.F=${global.resource.path}/fonts/TW-Sung-Plus-98_1.ttf
```

```
font.CNS11643.KAI.0=${global.resource.path}/fonts/TW-Kai-98_1.ttf
font.CNS11643.KAI.2=${global.resource.path}/fonts/TW-Kai-Ext-B-98_1.ttf
font.CNS11643.KAI.F=${global.resource.path}/fonts/TW-Kai-Plus-98_1.ttf
```

字型設定

主要的設定值有兩項：'default.font' 與 'default.font.size'，分別定義文件產出時預設使用的字型代碼與大小。預設範本使用「WINDOWS-細明體」輸出文字。

```
default.font=WindowsFont.MINGLIU
default.font.size=12
```

UNICODE 多字面

實際使用的字型檔案則以 `font.{字型代碼}.{UNICODE字面}` 進行定義。如下例中，`font.CNS11643.SUNG.0` 為全字庫宋體字第0字面，`font.CNS11643.SUNG.F` 為全字庫宋體字第15字面。

全字庫字型請自行至「政府資料開放平臺」<http://data.gov.tw/node/5961>下載

若使用完整UDE套件及相關設定方式，設定路徑可使用`${global.resource.path}`、`${resource.path}`等環境變數，讀入設定檔時會自動代換。

```
font.CNS11643.SUNG.0=${global.resource.path}/fonts/TW-Sung-98_1.ttf
font.CNS11643.SUNG.2=${global.resource.path}/fonts/TW-Sung-Ext-B-98_1.ttf
font.CNS11643.SUNG.F=${global.resource.path}/fonts/TW-Sung-Plus-98_1.ttf
```

多個字型檔案

如相同的字面設定有多組，則套件啟動時，會逐一測試各檔案路徑是否存在可用字型檔。像下例中把常用的 **WINDOWS** 標楷體字型所在目錄都做了設定，但只有實際存在的檔案會被引入，其它項目則被忽略。

```
font.WindowsFont.KAI.0 = C:/windows/fonts/kaiu.ttf
font.WindowsFont.KAI.0 = D:/windows/fonts/kaiu.ttf
font.WindowsFont.KAI.0 = C:/winnt/fonts/kaiu.ttf
font.WindowsFont.KAI.0 = D:/winnt/fonts/kaiu.ttf
font.WindowsFont.KAI.0 = ${global.resource.path}/fonts/kaiu.ttf
```

專案有使用自造字區者，也可以使用多字型檔案設定方式，把自造字檔案加到第二組。

```
font.NPP_FONT.MING.0=${global.resource.path}/font/TW-Sung-98_1.ttf
font.NPP_FONT.MING.0=${global.resource.path}/font/FNPMing.ttf
```

TTC 字型檔

TrueType Collections (TTC) 允許將多個字型合併為一個檔案。若使用 TTC 第二組以後的字型定義，如「新細明體」，可使用下列方式設定。

```
font.WindowsFont.NEW-MINGLIU.0 = C:/windows/fonts/mingliu.ttc,1  
font.WindowsFont.NEW-MINGLIU.2 = C:/windows/fonts/mingliub.ttc,1
```

單元測試

一般建議使用 `AbstractITextTestkit` 或 `AbstractITextTest` 為父類別，實作單元測試，因為有一些附加的測試特性可以選用。

AbstractITextTestkit

基礎的測試類別，主要功能有二：提供 `PDFDocumentManager` 以及依據測試 `METHOD` 名稱產出輸出檔案物件。建構子可傳入 `ITextTestConfig` 以定義測試設定(也可使用預設無參數設定)。

```
public class GSS0010Test extends AbstractITextTestkit {

    final GSS0010 gss0010 = new GSS0010();

    public GSS0010Test() {
        super(prepareConfig());
    }

    private static ITextTestConfig prepareConfig() {
        final ITextTestConfig config = new ITextTestConfig();
        config.setFixedBaseDir(".");
        config.setPath(UdeSystemPropertyItem.GLOBAL_RESOURCE_PATH, "${BASEDIR}");
        config.iTextConfigPath = "classpath:itext-config-test.properties";
        config.outputRoot = new File(FileUtils.getTempDirectory(), "__Report_Output");

        config.keepOutputFile = true;
        config.showMarginBorder = true;
        return config;
    }

    /** 產出檔名為：GSS0010Test/[GSS0010Test]PDF.pdf */
    @Test
    public void testPDF() {
        this.gss0010.generatePDF(getDocumentManager(), createFileByTestName(DocumentFormat.PDF));
    }

    /** 產出檔名為：GSS0010Test/[GSS0010Test]Excel.xlsx */
    @Test
    public void testExcel() {
        this.gss0010.generateExcel(createFileByTestName(DocumentFormat.EXCEL));
    }

}
```

以下是 `ITextTestConfig` 的可設定內容：

- `iTextConfigPath`: iText設定檔路徑，預設為 `classpath:itext-config-default.properties`。
- `outputRoot`: 指定輸出PDF的根目錄。預設為系統TEMP路徑下的__Report_SAMPLE子目錄。
- `keepOutputFile`: 執行完成後，是否保留檔案，預設為TRUE。但在 CI 環境中，應以環境變數 `keepOutputFile` 設定為 FALSE。
- `showMarginBorder`: 輸出的範例PDF，是否額外輸出邊界虛線，預設為TRUE。此項設定可以有效幫助開發人員調整版面設定。如果要產出給客戶做需求確認時，可以視需求設為FALSE。
- UDE 套件設定
 - `setFixedBaseDir(String)`: 在單一測試中，抽換所使用的系統環境變數 `BASEDIR` 值。
 - `setPath(UdeSystemPropertyItem, String)`: 在單一測試中，指定系統環境路徑變數值。

AbstractITextTest

TODO: 細部說明待補

SAMPLE

主要使每一個測試案例可以只撰寫 `generatePDF(...)` 所需的內容。除非展示文件共通特性，才另行撰寫獨立文件 **Generator** 類別。

TODO : 細部說明待補

概述

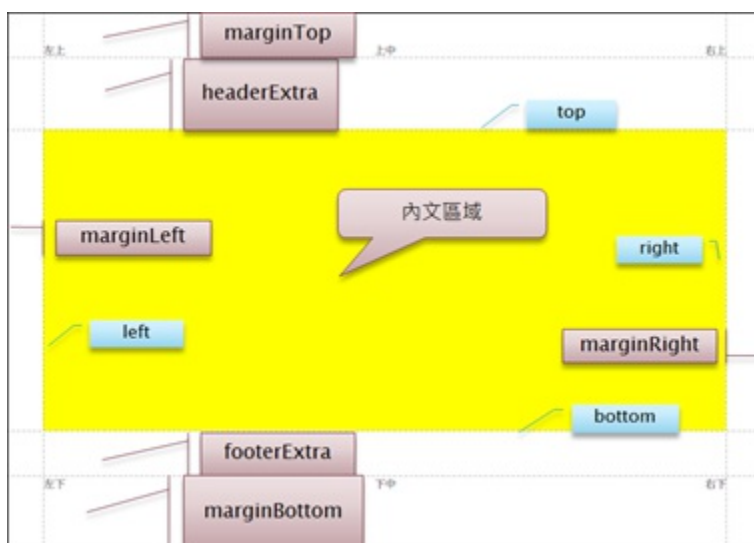
如PDF、WORD之類的文件，是複雜度最高的一種的產出格式。

一份完整的文件，包含封面、目錄、內頁等組成部份。而一般報表類型產出，就只需處理內頁。

文件各組成部分的主要差異，是不同的頁面版型及頁碼原則。例如封面頁通常沒有頁碼、目錄頁的頁碼可能會與內頁分別計算。

頁面版型結構

在此統一定義頁面組成及各元素輸出的基準位置，以下用橫式A4頁面為例說明。



頁首 PageHeader

圖中上方區域，每頁重複的部分，常用於輸出公用資訊如製表時間、報表/文件名稱、頁碼等。一般 API 工具定義這部分空間的高度為 **top**(圖中藍底標注值)。實際輸出頁首內容時，通常會加上一段留白以保持美觀或供輸出其它內容運用。UDE-Report 中，定義頁首內容實際輸出的底端到頁面上緣的距離為 **marginTop**；留白高度為 **headerExtra**，預設為 6 pixel。

頁尾 PageFooter

圖中下方區域，每頁重複的部分，常用於輸出每頁資訊如頁碼、註腳。一般 API 工具定義這部分空間參考值為 **bottom**(圖中藍底標注值)。實際輸出頁尾內容時，通常會加上一段留白以保持美觀或供輸出其它內容運用。UDE-Report 中，定義頁首內容實際輸出的底端到頁面上緣的距離為 **marginBottom**；留白高度為 **footerExtra**，預設為 6 pixel。

內文區域

圖中由藍底的4個定位值 (**top**、**bottom**、**left**、**right**) 所框起來的黃色區域，即是內文區。

表格 (Table)

產出類型為報表資料時的主體內容。以文件輸出表格時，顯示格式的彈性會較試算表為自由。

- 表首/標題 **TableHeader**

與表格相關的公用資訊，如輸出清單表格時，此處可顯示條件摘要。可以在每一頁重複出現，也可以只在一開始輸出一次。

與文件頁首略有不同的地方是：文件頁首位置固定；而表格標題則是跟著表格位置輸出，同一張表輸出多次，就有多個。

表格標題輸出：

表格一標題		
TEST1	TEST1	TEST1
TEST1	TEST1	TEST1
TEST1	TEST1	TEST1

表格二標題		
TEST2	TEST2	TEST2
TEST2	TEST2	TEST2
TEST2	TEST2	TEST2

- 表頭 **ColumnHeader**

與欄位資料對應的標題列，通常每一頁重複出現。

但輸出為試算表格式時，則只在一開始輸出一次。

- 表身

- **ColumnBody**：資料內容。
- **GroupHeader**
- **GroupFooter**

- 表尾 **TableFooter**

也可定義是否每頁重複(內容可變)，或只出現在最末。

PDF (itext)

以 **itext** 產製 PDF 文件的基本步驟為建立 **Document**、**PdfWriter**物件，並使用相關 **methods** 進行內容輸出。**UDE-Report** 套件已包裝所有文件開關流程。

使用基本 **API**

```
@Test
public void testCreatePDF() throws IOException {
    final PDFDocumentManager documentManager = getDocumentManager();
    final File file = createFileByTestName(DocumentFormat.PDF);
    try (PDFDocument pdfDocument = documentManager.createPDFDocument(file, PageSize.A5)) {
        pdfDocument.writeText("TEST");
    }
}
```

繼承 **AbstractPDFGenerator**

繼承並實作 `generatePDFContent(PDFDocument pdfDocument)`，可以得到更多關於LAYOUT、HEADER、FOOTER、分節的控制支援。最後呼叫 `generatePDF(...)` 即可產出PDF：

```
void generatePDF(PDFDocumentManager, File)
void generatePDF(PDFDocumentManager, OutputStream)
```

MS Word

UDEReport 尚未支援 **WORD** 格式輸出。

概述

- 如 Excel
- Grid-Table
- 頁首、頁尾樣式受限

Excel

負責產製Excel的類別是ExcelDocument，使用createXLS／XLSX可分別得到用於 Excel 97~2003 或Excel2007版本的對應物件。製表時，由 ExcelDocument createSheet() 得到ExcelSheet，然後利用appendCell() / appendFormulaCell() 加入欄位。最後呼叫ExcelDocument.close() 即可完成檔案輸出。

ODS

尚未支援。

以 `org.apache.commons.csv.CSVPrinter` 輸出

概述

因為各種輸出類型的天生限制，各底層API對頁面控制的處理支援度自然有所不同。「文件」類型，是對於頁面控制支援度最齊全的格式。

- PDFDocument

```
setPageSize(Rectangle)      // 頁面大小與方向
setLayoutInfo(LayoutInfo)   // 邊界留白與頁首頁尾
setMarkers(List<? extends MarkInfo>) // 浮水印相關定義
```

- ExcelDocument - ExcelSheet

在EXCEL中，頁面控制原則上以 **sheet** 為單位設定。目前 **ude-reports** 對於 EXCEL 版面控制沒有做太多的包裝處理，有需要的話，可以透過 **getRealSheet()** 取得 **poi** 原生物件進行設定。

```
setPrintPageSize(PrintPageSize)
```

頁面大小

目前 PDF / Excel 皆支援設定頁面大小。唯PDF 格式的頁面定義較彈性，EXCEL 受限於 POI API，只有固定的選項可用。

- Sample_PageSize.java

```
@Test
public void test_PDF_A3() throws IOException {
    super.createPDF(pdfDocument -> {
        pdfDocument.setupPageSize(PageSize.A3);
        pdfDocument.writeText("A3");
    });
}

@Test
public void test_Excel_A3() {
    super.createExcel(excelDocument -> {
        final ExcelSheet<?> sheet = excelDocument.createSheet("sheet_a3");
        sheet.setPrintPageSize(PoiDefaultSize.A3);
    });
}
```

共通 Generator 介面

當同一種報表內容要用不同格式輸出時，通常會用共同的 **AbstractReport** 實作 **DocumentGenerator** 的任一介面，如 **PDFGenerator**、**ExcelGenerator**...

分節 **getSections()**

UDE-Report 可利用 **Section** 機制，組合多個 **DocumentGenerator** 輸出為同一檔案，見後文說明。

BaseLayoutInfo 邊界與頁首頁尾控制

見後文說明。

MultiFormatReportSupport

當文件產出類別同時提供多種格式選擇時，可加入實作 **MultiFormatReportSupport**，它依據以下兩個函式提供 **createExcelSheet(document)::ExcelSheet<?>**。

```
default String toExcelSheetName() {
    return "sheet";
};
default PrintPageSize toPrintPageSize() {
    final Rectangle pdfPageSize = getPageSize();
    return HELPER.lookupPringPageSize(pdfPageSize);
};
```

所以在第一章架構範例中的表樣別(GSS0010)裡，可以在建構子使用 **PageSize.A3** 參數，同時定義 **PDF** 及 **EXCEL** 產出的預設頁面大小。

- GSS0010.java

```
public GSS0010() {
    super(AllReports.GSS0010, PageSize.A3);
}

@Override
public void generatePDFContent(final PDFDocument pdfDocument) {
    pdfDocument.writeText("TEST-GSS0010");
}

@Override
public void generateExcelContent(final ExcelDocument<?, ?> document) {
    final ExcelSheet<?> sheet = super.createExcelSheet(document);
    sheet.appendCell(new ExcelPoint(0, 0), "TEST-GSS0010", new CellFormat(Border.BOX));
    sheet.setColumnWidth(0, 20);
}
```

-
- Sample_PageSize.java

```
@Test
public void test_MultiFormat() {
    final GSS0010 report = new GSS0010();
    super.createPDF(report);
    super.createExcel(report);
}
```

內容分節

分節控制，一般應用於輸出文件類型的檔案，可因應封面頁、目錄等部分版面配置不同的需求。另一個常見用途是組合不同報表做客制化輸出。

依文件類型特性，分節處理模式自然不同。像是 EXCEL 預設會以獨立 SHEET 輸出各節內容；CSV 預設直接接續輸出所有內容。若有其它較特別的需求，也可以自行覆寫 `xxxGenerator` 中的相關實作處理。

基本範例 (PDF)

下例 `PDFSection` 用於輸出基本頁面，產出指定大小頁面配上單一文字。

- `PDFSection.java`

```
// PDFSection is a 'PDFGenerator'.
public class PDFSection extends AbstractPDFGenerator {
    private String text;
    public PDFSection(final Rectangle pageSize, final String text) {
        super(pageSize);
        this.text = text;
    }
    @Override
    public void generatePDFContent(final PDFDocument pdfDocument) {
        pdfDocument.writeText(this.text, 28, DocumentAlign.CENTER);
    }
}
```

我們可以建立多個 `PDFSection` 實例，並組裝起來，使其依 1、2、2-1、2-2、2-3、3、4、5 的順序輸出。

組裝方式有二：

- 第一層的項目可以直接用 `AbstractPDFGenerator.of(PDFGenerator...)` 產出一個空的 ROOT PDF 文件包含指定內容。
 - 一般專案實作中，應該要自訂空白 ROOT 文件類別，以便處理專案所需的邏輯，如檔案置放原則等等...
- 第二層以後的項目可用 `AbstractPDFGenerator` 所定義的 `addSection()` method 加入。

- `Sample_Sections.java`

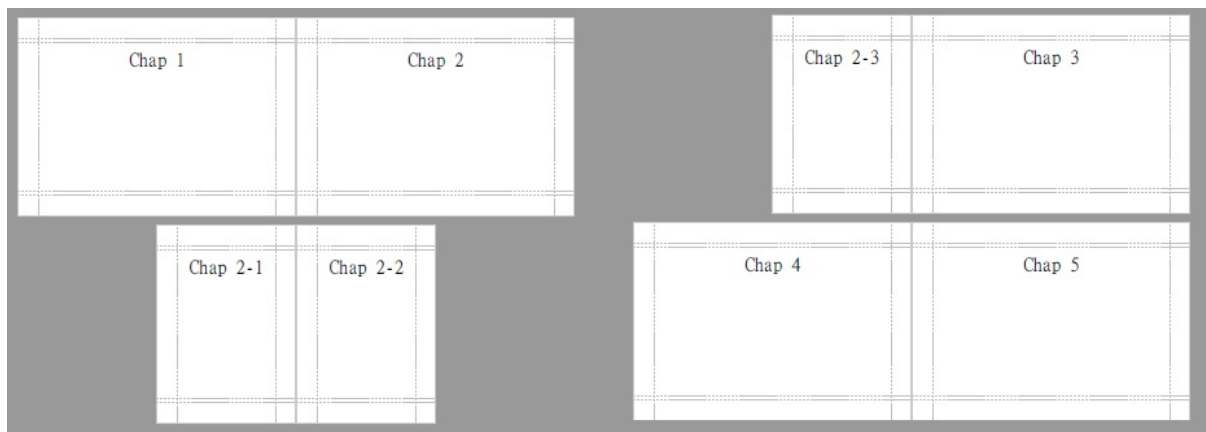
```
@Test
public void test_PDF_Section() {
    // 第一層項目
    final PDFSection s1 = new PDFSection(PageSize.A6.rotate(), "Chap 1");
    final PDFSection s2 = new PDFSection(PageSize.A6.rotate(), "Chap 2");
}
```

```

final PDFSection s3 = new PDFSection(PageSize.A6.rotate(), "Chap 3");
final PDFSection s4 = new PDFSection(PageSize.A6.rotate(), "Chap 4");
final PDFSection s5 = new PDFSection(PageSize.A6.rotate(), "Chap 5");
final AbstractDocumentGenerator root = AbstractPDFGenerator.of(s1, s2, s3, s
4, s5);
// 第二層以後的項目
s2.addSection(new PDFSection(PageSize.A7, "Chap 2-1"));
s2.addSection(new PDFSection(PageSize.A7, "Chap 2-2"));
s2.addSection(new PDFSection(PageSize.A7, "Chap 2-3"));
// CREATE PDF
super.createPDF(root);
}

```

執行結果如下，使用分節特性，可以做出各分節頁面版型不同的效果。



Excel 分節 (多SHEET)

接著拿實作 MultiFormatReportSupport 的 GSS0010 測試。在產出 EXCEL 時，各分節會以不同 SHEET 呈現。

- Sample_Sections.java

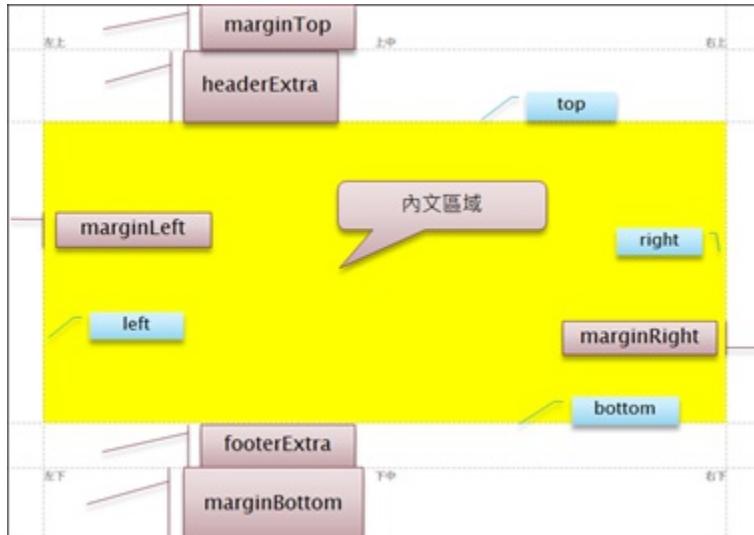
```

@Test
public void test_Excel_Section() {
    final GSS0010 s1 = new GSS0010();
    final GSS0010 s2 = new GSS0010("S1");
    final GSS0010 s3 = new GSS0010("S2");
    final DocumentGenerator root = AbstractSampleReport.of(s1, s2, s3);
    super.doDocument(root, DocumentFormat.EXCEL);
    super.doDocument(root, DocumentFormat.PDF);
}

```


邊界控制

- 適用於 PDF / EXCEL



再回顧一下基本概念中看到的「頁面版型結構」，使用 **BaseLayoutInfo** 可定義其中的定位資訊。

單位為 **pixel**，可以使用 **LengthUnit** 的 **trans()** 函式進行換算，如下例為公分、公厘、英吋不同單位設定結果。

(邊界虛線以測試套件的 **showMarginBorder=TRUE** 輸出。)

- Sample_Page_Margin.java

```
private LayoutInfo createLayout() {
    // ! 定義四周邊界大小.
    final float marginLeft = LengthUnit.CM.trans(2.54f); // 左方以 公分為單位，合 1
    英吋/72pixel
    final float marginRight = 36; // 右方以 pixel 為單位，合
    0.5英吋
    final float marginTop = LengthUnit.MM.trans(12.7f); // 上方以 公厘為單位，合 0.
    5英吋/36pixel
    final float marginBottom = LengthUnit.INCH.trans(1); // 下方以英吋為單位， 1英吋
    final LayoutInfo layoutInfo = new LayoutInfo(marginLeft, marginRight, marginTo
    p, marginBottom);
    // ! 定義上下頁首尾間距.
    layoutInfo.setHeaderExtra(36);
    layoutInfo.setFooterExtra(72);
    return layoutInfo;
}

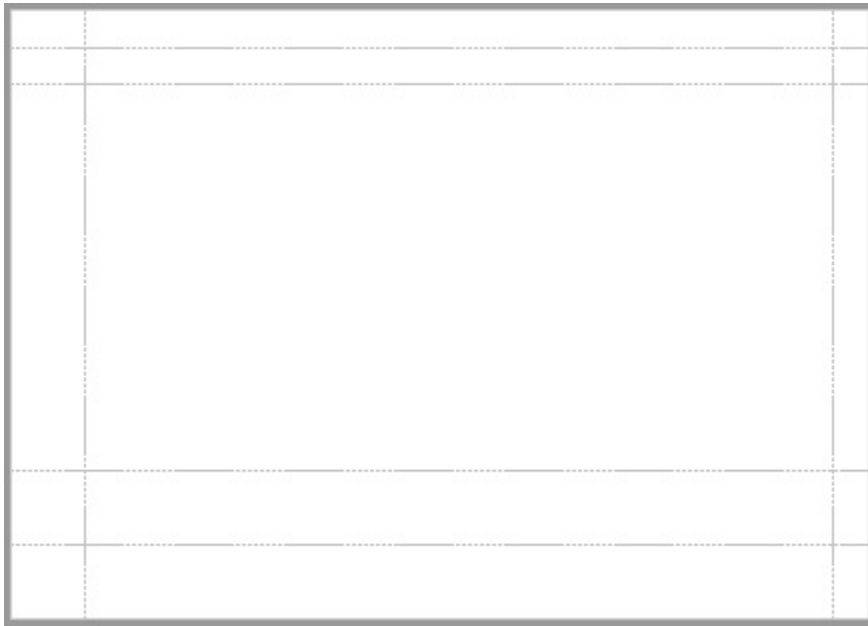
@Test
public void test_PDF_Margin() {
    super.createPDF(pdfDocument -> {
```

```

        pdfDocument.setupPageSize(PageSize.A4.rotate()); // ! 定義頁面大小
        pdfDocument.setLayoutInfo(createLayout());        // ! 設定版面資訊
    });
}

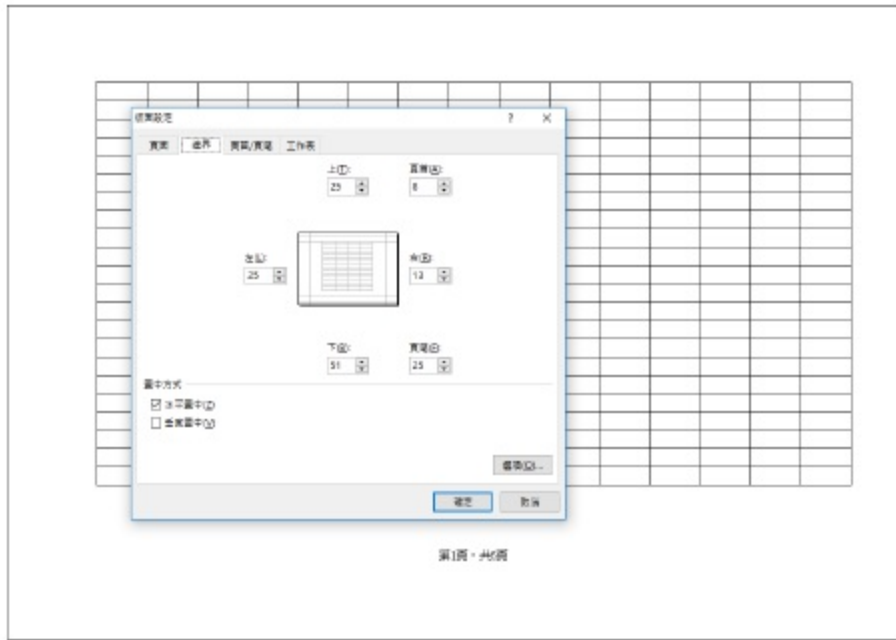
@Test
public void test_Excel_Margin() {
    super.createExcel(excelDocument -> {
        final ExcelSheet<?> sheet = excelDocument.createSheet("sheet");
        sheet.setPrintPageSize(PoiDefaultSize.A4L);    // ! 定義頁面大小.
        sheet.setLayoutInfo(createLayout());            // ! 設定版面資訊
        for (int i = 0; i < 1000; i++) {
            sheet.appendCell(new ExcelPoint(i / 20, i % 20), "");
        }
    });
}

```



- PDF

- EXCEL：相較Excel中的版面設定參數，可以看出部分定義不同，已由 UDE-Report 進行轉換



處理。

換頁控制

- 適用於 PDF

要手動對PDF進行換頁，請呼叫 `PDFDocument.newPage()`，若當前頁面還沒有實際內容輸出時，並不會真的換頁。所以如果要輸出空白頁面，請在兩次換頁間插入一次 `writeText("")`。

```
@Test
public void test_newPage() {
    super.createPDF(pdfDocument -> {
        final LayoutInfo layoutInfo = new LayoutInfo();
        layoutInfo.setHeader(ItemPosition.CenterFooter, new PageHeaderCH(12));
        pdfDocument.setLayoutInfo(layoutInfo);
        pdfDocument.setPageSize(PageSize.A8.rotate());
        pdfDocument.writeText("P1"); // 一開始在 P1
        pdfDocument.newPage(); // 進入 P2
        pdfDocument.newPage(); // 連續兩次呼叫兩次 newPage() 不會有空白頁
        pdfDocument.writeText("P2");
        pdfDocument.newPage(); // 進入 P3
        pdfDocument.newPage();
        pdfDocument.newPage();
        pdfDocument.newPage(true); // 強制換頁後，才進入 P4
        pdfDocument.writeText("P4. 我是第四頁");
        pdfDocument.newPage(); // 後續未輸出內容，P5 不會產生。
    });
}
```




輸出內容後，如果要確認是否造成換頁，可以用`pdfDocument.isPageChanged()`、`pdfDocument.isNewPageBegin()` 兩個Method 進行判斷。

- `PageChanged` 為真：有觸發換頁事件。
- `NewPageBegin`為真：新頁面還沒有輸出任何內容。

```
@Test
public void test_forceNewPage() {
    super.createPDF(pdfDocument -> {
        final LayoutInfo layoutInfo = new LayoutInfo();
        layoutInfo.setHeader(ItemPosition.CenterFooter, new PageHeaderCH(12));
        pdfDocument.setLayoutInfo(layoutInfo);
        pdfDocument.setPageSize(PageSize.A8.rotate());
        // !
        Assert.assertFalse("一開始未換頁", pdfDocument.isPageChanged());
        Assert.assertTrue(pdfDocument.isNewPageBegin());
        // !
        pdfDocument.writeText("P1");
        Assert.assertFalse(pdfDocument.isPageChanged());
        Assert.assertFalse(pdfDocument.isNewPageBegin());
        // !
        pdfDocument.newPage(true);    // 強制換頁
        Assert.assertTrue("因為 newPage() 觸發換頁事件", pdfDocument.isPageChanged());
    });

    Assert.assertTrue("新頁面還沒有輸出任何內容，但關檔時強制輸出新頁。", pdfDocument
```

```
.isNewPageBegin());  
    });  
}
```



頁面大小與邊界，在文件生成的過程中是可以改變的。當目前頁面已有內容輸出時，頁面大小變動會在下一頁才生效。

但邊界變動可能會導致部分頁首頁尾輸出與預期不符，使用時請自行確認當下文件輸出狀態。換言之，建議把**PageSize**與**LayoutInfo**的異動緊接在**newPage()** 之後進行。

但一般不建議在文件中變動頁面大小，這類文件可能會對使用者列印輸出或做其它處理造成困擾。

頁首、頁尾控制

頁首、頁尾同樣經由 **BaseLayoutInfo** 設定。

但因 **Excel** 的支援性問題，**BaseLayoutInfo** 只可設定基本的文字或頁面輸出。

若用 **LayoutInfo** 輸出 **PDF**，則有更多彈性的樣式可以選擇。

以下是設定頁首、頁尾時的兩個主要參數 **ItemPosition** 及 **RepeatMode**：

- **BaseLayoutInfo.java**

```
public class BaseLayoutInfo {
    /** 設定指定位置上的 HEADER. */
    void setHeader(ItemPosition position, Header header) {}
    /** 增加指定位置上的 HEADER. */
    void addHeader(ItemPosition position, Header header) {}
    /** 清除指定位置上的 HEADER. */
    void removeHeaders(ItemPosition position) {}
}
public interface Header {
    default RepeatMode getRepeatMode() { return RepeatMode.ALL; }
}
```

在設定 **Header** 時，必須指定 **ItemPosition**，也就是相同的 **Header** 可以被放置在文件上的任何位置。而 **Header** 介面本身要求回傳 **RepeatMode**，由此決定該 **HEADER** 在哪些頁次會被輸出。**getRepeatMode()** 的預設回傳為 **ALL**，也就是在所有頁次都會出現。

定位點 (ItemPosition)

輸出位置由 **position** 指定，依左中右／上下組合，共有 8 個位置：

- 文字輸出於左邊時置左；右邊時置右；中間時置中。
- 輸出在上方時，會靠下對齊 **marginTop** 定位點。
- 輸出在下方時，會靠上對齊 **marginBottom** 定位點。
- **PageCenterFooter** / **PageCenterHeader**

若預留空間不足，則輸出內容會超出頁首、尾範圍。

- **Sample_PageHeader_BasicText.java :: test_BASELINE**

```
final LayoutInfo layoutInfo = new LayoutInfo();
layoutInfo.setTextHeader(ItemPosition.LeftHeader, text, 14);
layoutInfo.setTextHeader(ItemPosition.CenterHeader, text, 12);
layoutInfo.setTextHeader(ItemPosition.RightHeader, text, 8);
layoutInfo.setTextHeader(ItemPosition.LeftFooter, text, 8);
layoutInfo.setTextHeader(ItemPosition.CenterFooter, text, 12);
```

```
layoutInfo.setTextHeader(ItemPosition.RightFooter, text, 14);
pdfDocument.setLayoutInfo(layoutInfo);
```



- PageCenter VS Center

TODO

重複模式 (RepeatMode)

Name	說明
ALL	每頁出現
FIRST_PAGE	只在文件的第一頁輸出 (不論該頁是否計算頁數)
CONTENT_PAGES	於頁次計數不為0的頁面輸出 (ONLY-FOR-PDF)
COVERAGE_PAGES	於頁次計數為0的頁面輸出 (ONLY-FOR-PDF)
ODD_PAGES	於奇數頁次輸出
EVEN_PAGES	於偶數頁次輸出

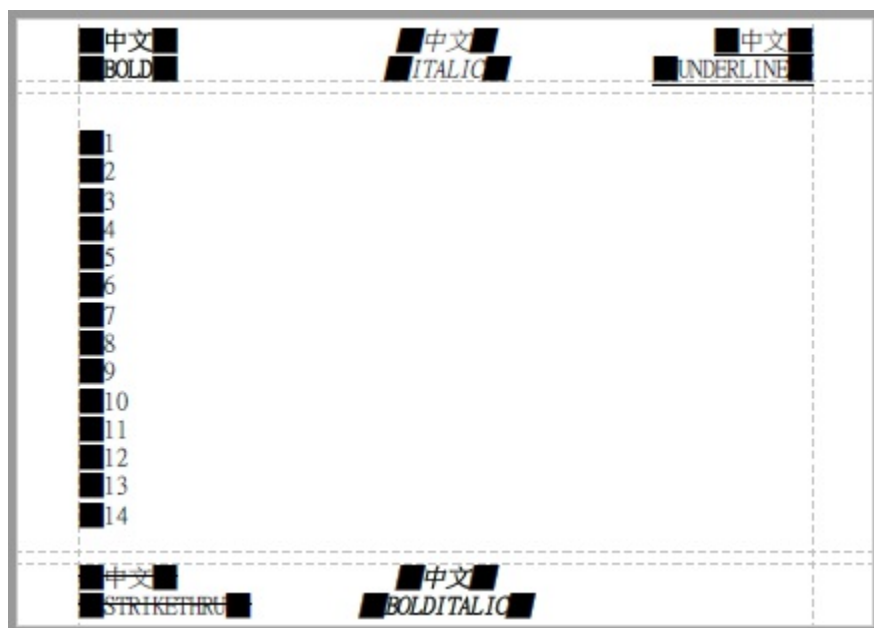
基本文字輸出

`set/addTextHeader` 可以輸出固定文字內容並且指定字體大小，上例即使用此方法設定頁首、頁尾。基本文字輸出也可以控制字體樣式，如下例。注意 **UNDERLINE** 樣式，應用在頁首時，可能會佔用 `HeaderExtra` 的空間 (兩條虛線之間)。

```
addTextHeader(ItemPosition, RepeatMode, String, int)
addTextHeader(ItemPosition, RepeatMode, String, int, FontStyle)
addTextHeader(ItemPosition, String, int)
addTextHeader(ItemPosition, String, int, FontStyle)
setTextHeader(ItemPosition, RepeatMode, String, int)
setTextHeader(ItemPosition, RepeatMode, String, int, FontStyle)
setTextHeader(ItemPosition, String, int)
setTextHeader(ItemPosition, String, int, FontStyle)
```

- `Sample_PageHeader_BasicText.java :: test_fontStyle`

```
final LayoutInfo layoutInfo = new LayoutInfo();
layoutInfo.setTextHeader(ItemPosition.LeftHeader, String.format(text, "BOLD"),
12, FontStyle.BOLD);
layoutInfo.setTextHeader(ItemPosition.CenterHeader, String.format(text, "ITALI
C"), 12, FontStyle.ITALIC);
layoutInfo.setTextHeader(ItemPosition.RightHeader, String.format(text, "UNDERL
INE"), 12, FontStyle.UNDERLINE);
layoutInfo.setTextHeader(ItemPosition.LeftFooter, String.format(text, "STRIKET
HRU"), 12, FontStyle.STRIKETHRU);
layoutInfo.setTextHeader(ItemPosition.CenterFooter, String.format(text, "BOLDI
TALIC"), 12, FontStyle.BOLDITALIC);
pdfDocument.setLayoutInfo(layoutInfo);
```



- **TODO : EXCEL** 支援及範例

基本頁碼輸出

`set/addPagingHeader` 可以輸出目前頁次，並指定字體大小。

主要參數為 `PagingPattern`，以 `PagingItem` 定義使用的頁次資訊；`getPrefix()`、`getConjunction()`、`getSuffix()` 定義前後綴及連接文字。

若要更客制化的輸出多種頁次資訊，目前只支援 PDF 格式，需繼承 `AbstractPagingHeader` 實作。

PagingItem	說明	預設英文格式 (PageHeaderEN)	預設中文格式 (PageHeaderZH)
PAGE	文件頁次	Page {p}	第{p}頁
TOTAL_PAGES	文件頁數	Total pages:{tp}	共{tp}頁
BOTH	文件頁次 + 文件頁數	Page {p} of {tp}	第{p}頁，共{tp}頁
SECTION	目前節次	{s}	第{s}節
PAGE_IN_SECTION	目前節內頁次	Page {sp}	第{sp}頁
SECTION_AND_PAGE	目前節次 + 目前節內頁次	Page {s} - {sp}	第{s}~{sp}頁
SECTION_PAGES	各節總頁數	Total pages:{tsp}	共{tsp}頁

- Sample_PageHeader_BasicPage :: test_BasicPaging

```
final PDFSampleContent setting = pdfDocument -> {
    pdfDocument.setPageSize(PageSize.A5.rotate());
    final LayoutInfo layoutInfo = new LayoutInfo();
    layoutInfo.setPagingHeader(ItemPosition.LeftHeader, PagingHeaderEN.PAGE, 14)
    ;
    layoutInfo.setPagingHeader(ItemPosition.CenterHeader, PagingHeaderEN.BOTH, 14)
    );
    layoutInfo.setPagingHeader(ItemPosition.RightHeader, PagingHeaderEN.TOTAL_PA
    GES, 14);
    layoutInfo.setPagingHeader(ItemPosition.LeftFooter, PagingHeaderZH.PAGE, 10)
    ;
    layoutInfo.setPagingHeader(ItemPosition.CenterFooter, PagingHeaderZH.BOTH, 10)
    );
    layoutInfo.setPagingHeader(ItemPosition.RightFooter, PagingHeaderZH.TOTAL_PA
    GES, 10);
    pdfDocument.setLayoutInfo(layoutInfo);
};
super.createPDF(setting.andThen(this::outputRepeatText));
```


Page 1-1	1 of 4			2 of 4	Page 1-2
Total pages:6	Sec. 1	Section Pages:4		Total pages:6	Sec. 1 Section Pages:4
Page 1-3	3 of 4			4 of 4	Page 1-4
Total pages:6	Sec. 1	Section Pages:4		Total pages:6	Sec. 1 Section Pages:4
Page 2-1	1 of 2			2 of 2	Page 2-2
Total pages:6	Sec. 2	Section Pages:2		Total pages:6	Sec. 2 Section Pages:2

- **TODO : EXCEL 支援及範例**

PDF 格式支援

多文字格式

表格輸出

若有需要，可用 `DocumentFormat` 參數做判斷處理。

頁次管理

- **TODO** : 待完善實作

浮水印

- 適用於 **PDF**

浮水印與騎縫章是相似的概念，都是輸出在文件底部的圖片或文字，差別在於騎縫章會把一張圖片切為兩半，分別輸出在不同頁面的首尾。兩者的定義方式都是產出對應的**MarkInfo**，在產生**PDFDocument**時傳入**List<? extends MarkInfo> markInfos**。或在**PDFGenerator**實作**prepareMarkInfos()**

套件內建的浮水印、騎縫章都統一使用**PropertiesMarkInfo**進行定義，依不同形式的浮水印、騎縫章，各自有對應的**MarkmakerProperties**實作子類別。

文字浮水印

文字型態浮水印資訊的實作類別是**TextWatermarkProperties**，預設建構子為輸出文字，預設字型使用**PDFDocument**的定義，所以通常必要再設定它的**FontType**及**FontSize**。不特別指定角度的話，預設依內頁範圍的左下-右上對角輸出。

圖片浮水印

圖片浮水印資訊的實作類別是**ImageWatermarkProperties**，預設建構子為**filePath**。若圖檔資源隨同**JAR**檔一並發布，則可用**classpath:\${resource-path}**格式指定。

文字內容

- 適用於 PDF

本章主要針對 PDF 輸出文字內容進行說明，因為對 Excel 文件而言，所有內容皆為表格輸出。一般文件的說明文字，如果沒有特別排版需求，可以直接使用文字輸出模式處理。

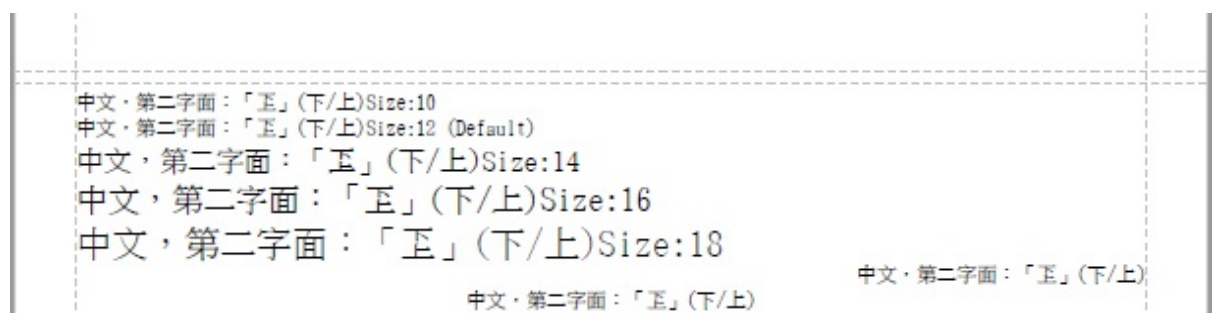
- 文字格式相關概念，原則上也適用於表格輸出。
- 若要精細控制文字輸出位置及樣式，一般套用表格輸出會比較簡單。

基本輸出

`writeText(...)` 提供基本輸出文字功能，可設定字型大小及對齊方式。

預設的字型大小為 `itext-config.properties` 中的「`default.font.size`」，也可經由 `pdfDocument.setFontSize()` 重新設定。基本輸出的預設行高為 1.25 倍字高。因為預設使用新細明體，所以可以顯示第二字面文字(字型檔:mingliub.ttc)。

```
@Test
public void test_basicText() {
    super.createPDF(pdfDocument -> {
        pdfDocument.setFontSize(12); // set Default Size
        final String text = "中文，第二字面：「丕」(下/上)";
        pdfDocument.writeText(text + "Size:10", 10);
        pdfDocument.writeText(text + "Size:12 (Default)");
        pdfDocument.writeText(text + "Size:14", 14);
        pdfDocument.writeText(text + "Size:16", 16);
        pdfDocument.writeText(text + "Size:18", 18);
        pdfDocument.writeText(text, 10, DocumentAlign.RIGHT);
        pdfDocument.writeText(text, 10, DocumentAlign.CENTER);
    });
}
```



預設字型調整

文件預設字型可使用`setFontInfo(...)`調整。可參考範例說明，但一般不會頻於設定文件預設字型。

ParagraphBuilder

特殊格式的文字，如粗體、變色，可使用`ParagraphBuilder`完成。由 `pdfDocument` 建立 `paragraphBuilder`後，即可逐一用`addText(text, fontInfo)`加入文字區塊，最後以`appendMe()`進行輸出。`ParagraphBuilder`，也可設定*iText*原生`Paragraph`物件中的各項設定值，如對齊、縮排等等...

在表格欄位中，若要輸出混合格式的文字，原理也與 `ParagraphBuilder` 相同。

預設字型調整

文件預設字型可使用`setFontInfo(...)`調整。

首先取得指定字體的`CHTFontFactory`，用以建立想要產出的字型資訊（`FontInfo`），可設定大小、Style、顏色、底色。但一般不會頻於設定文件預設字型。

```
pdfDocument.writeText("中文，第二字面：「" + "" + "」(下/上)");  
final CHTFontFactory kaiFactory = CHTFontFactories.INSTANCE.getFactory(WindowsFont  
.KAI);  
pdfDocument.setFontInfo(kaiFactory.createFontInfo(10, Color.BLUE));  
pdfDocument.writeText("轉成標楷體/藍色字，無第二字面可供顯示。");  
pdfDocument.writeText("中文，第二字面：「" + "" + "」(下/上)");  
pdfDocument.writeText("中文，第二字面：「" + "" + "」(下/上)", 18);
```

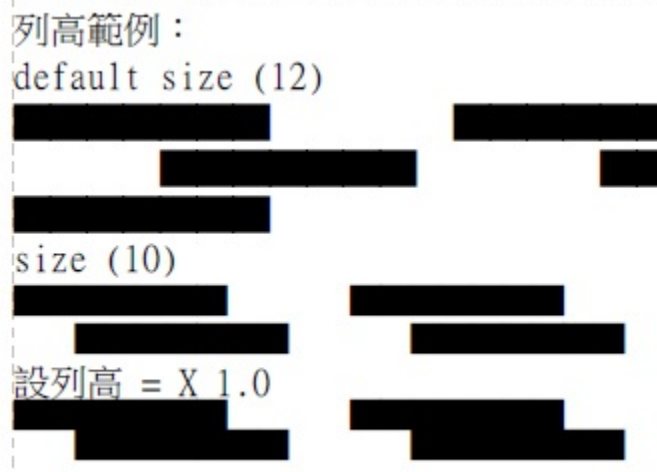
中文，第二字面：「𠂇」(下/上)
轉成標楷體/藍色字，無第二字面可供顯示。
中文，第二字面：「」(下/上)

中文，第二字面：「」(下/上)

由上例可看出，若使用含非0字面字型檔的字型，則本套件支援非0字面的中文輸出，一般直接使用iText輸出的範例程式是未做此項處理的。但若使用的字型不包含非0字面字型檔，則非0字面文字自然也無法顯示，同時不會佔用輸出空間。

列高設定

```
pdfDocument.setFontSize(12); // set Default Size
pdfDocument.writeText("列高範例：");
// !
pdfDocument.writeText("default size (12)");
pdfDocument.writeText(toBlockText(7, 5));
// !
pdfDocument.writeText("size (10)");
pdfDocument.writeText(toBlockText(7, 4), 10);
// !
pdfDocument.writeText("設列高 = X 1.0");
final FontInfo font10 = pdfDocument.getFontInfo().asSize(10);
pdfDocument.paragraphBuilder() //
    .addText(toBlockText(7, 4), font10) // 字型 10
    .setMultipliedLeading(1F) // 設定列高 = X 1.0
    .appendMe();
```



列高範例：

default size (12)

size (10)

設列高 = X 1.0

混合字型輸出

```
final CHTFontFactory kaiFactory = CHTFontFactories.INSTANCE.getFactory(WindowsFont
.KAI);
final FontInfo redFont = pdfDocument.getFontFactory().createFontInfo(10, FontStyle
.STRIKETHRU, Color.RED);
final FontInfo kaiFont = kaiFactory.createFontInfo(10, FontStyle.UNDERLINE, Color.
BLUE, Color.LIGHT_GRAY);
final ParagraphBuilder builder = pdfDocument.paragraphBuilder();
builder.addText("中文，第二字面：") //
    .addText("", redFont) //
    .addText("(下/上)", kaiFont) //
    .appendMe();
```

中文，第二字面：至(下/上)

UDE Report 以 CHTFontFactory 管理單一字型。

表格 (Table)

- 適用於 PDF / EXCEL

CellDataSource

CellFormat

本套件以 **CellFormat** 來定義個別欄位的顯示外觀，此物件可適用於 PDF 及 Excel 表格，唯部分性質可能在兩種文件格式中的支援度不同。於「表格描述定義」中，同樣使用 **CellFormat** 設定格式外觀。「表格欄位樣式」一節會再詳細說明。

表格描述定義

後面章節會介面 UDE-Report 的「表格描述定義」，可以針對 **JavaBean / Map** 形式資料來源，以相同的 **Table Metadata** 轉換為不同格式(PDF/Excel/CSV)輸出，方便客戶要求變換報表格式時處理。

本章介紹製表所用的基礎API，理論上所有的表格皆可以用基礎繪製，但是可使用「表格描述定義」時，還是建議盡量使用，以簡化程式開發。

基礎API 適用的應用情境包含格式複雜的申請書、跨頁的大型表格、格式簡單的名條、頁首頁尾特殊排版。

PDF

本套件主要使用 **iText** 中功能較完整的 **PdfPTable** 做為底層表格輸出元件。但使用時，應使用 Ude-Report 所包裝的 **TableiText** 類別操作。

TableiText 是 **PdfPTable** 的包覆類別，它提供中文處理能力、樣式簡易設定、跨欄、跨列與 **InnerTable** 相關操作，也可取得原生 **PdfPTable** 物件進行更複雜的操作。

TableiText 可使用 **PDFDocument** 的相關函式建立，當所有欄位加入表格後，再呼叫 **appendMe()** 進行輸出。如欲輸出於頁面上的指定位置，也可以使用「定位輸出」的相關工具類別函式處理，但是表樣的呈現彈性會較差。

Excel

ExcelSheet 類別已包裝常用的 **POI Excel** 輸出的相關功能。

```
appendCell(ExcelPoint, ExcelPoint, Object, CellFormat)
appendCell(ExcelPoint, Object, CellFormat)
```

```
appendCell(Point, Object, CellFormat)
appendCell(Point, Point, Object, CellFormat)
appendFormulaCell(ExcelPoint, String, CellFormat)
setPrintPageSize(PrintPageSize)
setBorder(ExcelPoint, ExcelPoint, Border)
setColumnWidth(int, float)
setColumnWidthInPixel(int, float)
setRowHeight(int, float)
```

基本表格輸出

PDF：建立 **TableText**

createTable(float widthPercentage, int numColumns)

等寬表格，共有 numColumns 欄，並指定表格寬度佔頁面比例 widthPercentage。

createTable(float widthPercentage, float[] widths)

指定欄寬比例widths，並指定表格寬度佔頁面比例 widthPercentage。

createTable(LengthUnit unit, float[] realWidths)

指定度量單位，並直接指定各欄的實際寬度。若總計寬度超過頁面大小，則自動以滿版分配比例顯示。

但用PaintTool公用類別輸出時，仍顯示為實際大小。處理特殊頁首頁尾時，有可能會使用此建構方式。

PDF 表格設定

TableText 繼承 PdfPTableWrapper，其中包含以下 delegate 函式可操作原生 PdfPTable，控制輸出位置及欄位呈現設定等等。

- setExtendLastRow(boolean)
- setFooterRows(int)
- setHeaderRows(int)
- setHorizontalAlignment(DocumentAlign)
- setHorizontalAlignment(int)
- setKeepTogether(boolean)
- setSkipFirstHeader(boolean)
- setSkipLastFooter(boolean)
- setSpacing(double)
- setSpacingAfter(double)
- setSpacingAfterInCM(float)
- setSpacingBefore(double)
- setSpacingBeforeInCM(float)
- setSplitLate(boolean)
- setSplitRows(boolean)

增加欄位內容. **ADD CELL**

TableText 所提供的 addCell(...) 函式，其參數依型別分類，排列組合而成

- 內容
 - `String text / CellDataSource source`：二選一，欄位的内容文字。
 - `SubPhrase... subPhrases`：`varargs` 必定在最後一項。
- 外觀
 - `CellFormat cellFormat`：本欄樣式。
 - `int colspan`：跨欄位數。

```

addCell(String, SubPhrase...)
addCell(String, CellFormat, int, SubPhrase...)
addCell(String, CellFormat, SubPhrase...)
addCell(String, int, SubPhrase...)
addCell(SubPhrase...)
addCell(CellFormat, int, SubPhrase...)
addCell(CellFormat, SubPhrase...)
addCell(int, SubPhrase...)
addCell(CellDataSource, CellFormat, int, SubPhrase...)
addCell(CellDataSource, CellFormat, SubPhrase...)
addCell(CellDataSource, int, SubPhrase...)
addCell(CellDataSource, SubPhrase...)
addCell(PdfPCell) // ITEXT 原生METHOD

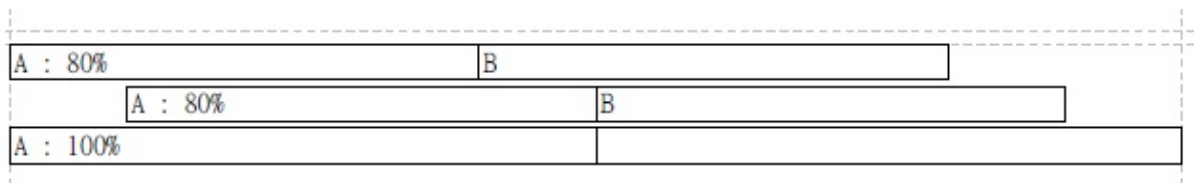
```

Excel

- 取得 `ExcelDocument<?, ?> document`。
- 呼叫 `document.createSheet(name)` 得到 `ExcelSheet<?>`。
- 輸出CELL 位置皆以 `ExcelPoint`定位，其行列計數由 0 起算。
 - `appendCell(ExcelPoint, Object)`
 - `appendCell(ExcelPoint, Object, CellFormat)`
- 有兩個 `ExcelPoint` 輸入時，表示合併儲存格。
 - `appendCell(ExcelPoint, ExcelPoint, Object, CellFormat)`
- 插入 EXCEL 公式需用專用 METHOD
 - `appendFormulaCell(ExcelPoint, String, CellFormat)`

指定表格寬度佔頁面比例

```
super.createPDF(pdfDocument -> {
    final TableiText table0 = pdfDocument.createTable(80, 2);
    table0.setHorizontalAlignment(DocumentAlign.LEFT);
    table0.addCell("A : 80%");
    table0.addCell("B");
    table0.appendMe();
    final TableiText table1 = pdfDocument.createTable(80, 2);
    table1.addCell("A : 80%");
    table1.addCell("B");
    table1.appendMe();
    final TableiText table2 = pdfDocument.createTable(100, 2);
    table2.addCell("A : 100%");
    table2.appendMe();
});
```



A : 80%	B
---------	---

A : 80%	B
---------	---

A : 100%	
----------	--

指定表格寬度佔頁面比例及欄寬權重

```
@Test
public void test_basicWidths() {
    super.createPDF(pdfDocument -> {
        final TableiText table1 = pdfDocument.createTable(100, 5);
        addWidthInfoCells(table1);
        table1.appendMe();
        final float[] widths = { 1, 2, 3, 4, 5 };
        final TableiText table2 = pdfDocument.createTable(100, widths);
        addWidthInfoCells(table2);
        table2.setSpacingBeforeInCM(0.5F);
        table2.appendMe();
    });
}

private static void addWidthInfoCells(final TableiText table) {
    final float[] widths = table.getWidths();
    final float totalWidth = UdeArrayUtils.sum(widths);
    for (int j = 0; j < widths.length; j++) {
        table.addCell("" + j);
    }
}
```

```

    for (final float width : widths) {
        table.addCell("" + width + "/" + totalWidth);
    }
}

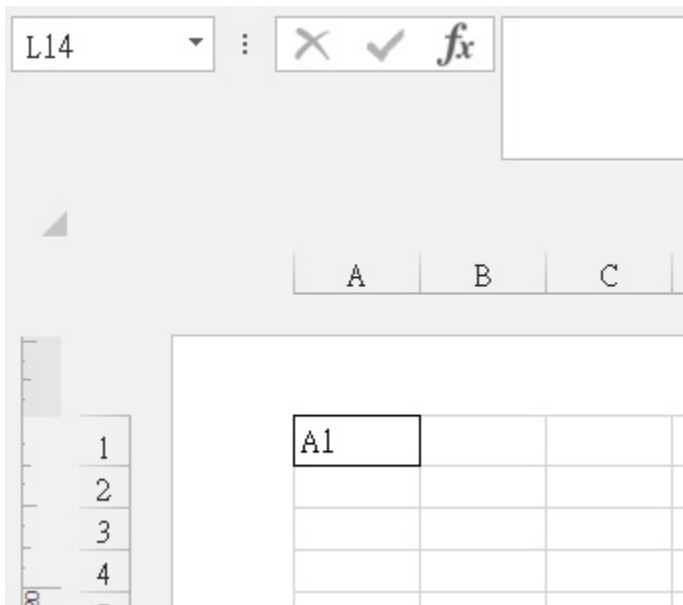
```

0	1	2	3	4
1/5	1/5	1/5	1/5	1/5

0	1	2	3	4
1/15	2/15	3/15	4/15	5/15

Excel Sample

```
@Test
public void test_basic() {
    super.createExcel(document -> {
        final ExcelSheet<?> sheet = document.createSheet("sheet1");
        sheet.appendCell(new ExcelPoint(0, 0), "A1");
    });
}
```



表格欄位樣式

本套件以 `CellFormat` 來定義個別 `Cell` 的顯示外觀。 `ExcelDocument` 及 `TableiText` 可用 `getDefaultFormat()` 取得並設定適用整個表格的預設外觀定義。 個別 `Cell` 則是在 `addCell` 時，傳入指定的 `CellFormat` 物件即可。

一般建議在輸出任何表格內容前，就設定 `defaultFormat` 完成。 若輸出欄位後，才異動 `defaultFormat`，雖不會影響到已輸出的欄位外觀，但維護性可能較差。

當表格中含有 `InnerTable` 時， `InnerTable` 的 `defaultFormat` 的未定義部分，會參考外表格的 `defaultFormat` 值。

以下逐一說明 `CellFormat` 的設定內容。

底色

- `setBackgroundColor(Color)`

邊框

- `setBorder(Border)`
- `setBorderWidth(Float)`

對齊

- `setAlignH(AlignH)`
- `setAlignV(AlignV)`

字型、輸出

- `setFontBold(Boolean)`
- `setFontSize(Integer)`
- `setFontType(FontType)`
- `setCellType(Celltype)`
- `setTextFormat(String)`

表格的字型預設樣式依 `AddCell` 當下， `pdfDocument` 的預設字型決定。 若要單獨設定，可使用 `getFontSize()`、`setFontType()`、`setFontBold()` 進行異動。 以上只支援基本粗體設定，若要用到複雜的格式，如斜體、底線、文字底色等等，請在 `AddCell` 時，使用 `(SubPhrase... subPhrases)` 逐一傳入文字區塊定義

大小

- `setMinHeightInCM(Float)`

留白

- `setPadding(double)`
- `setPaddingB(double)`
- `setPaddingH(double)`
- `setPaddingL(double)`
- `setPaddingR(double)`
- `setPaddingT(double)`
- `setPaddingV(double)`

底色

欄位底色可直接使用 `java.awt.Color` 設定 `backgroundColor`

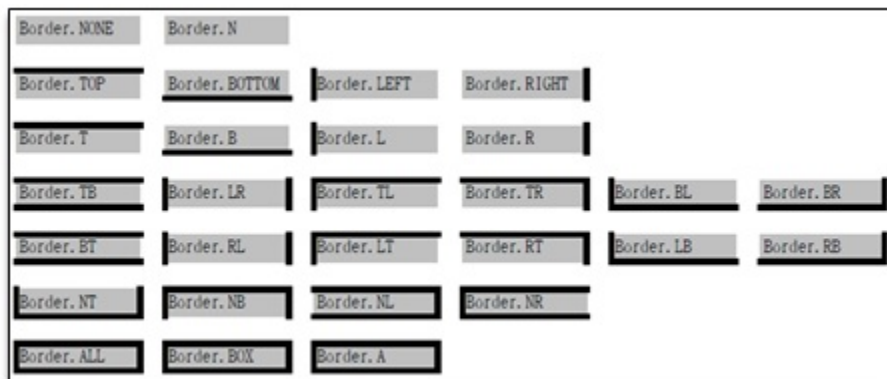
```
final TableiText table = pdfDocument.createTable(98, 16);
table.getDefaultFormat().setFontSize(8);
for (int r = 0; r < 255; r += 32) {
    for (int g = 0; g < 255; g += 32) {
        for (int b = 0; b < 255; b += 32) {
            final CellFormat cf = new CellFormat().setBackgroundColor(new Color(r,
            g, b));
            table.addCell(String.format("%X%X%X", r, g, b), cf);
        }
    }
}
table.appendMe();
```

0400	04020	04040	04060	04080	040A0	040C0	040E0	0500	05020	05040	05060	05080	050A0	050C0	050E0
0800	08020	08040	08060	08080	080A0	080C0	080E0	0A00	0A020	0A040	0A060	0A080	0A0A0	0A0C0	0A0E0
0C00	0C020	0C040	0C060	0C080	0C0A0	0C0C0	0C0E0	0E00	0E020	0E040	0E060	0E080	0E0A0	0E0C0	0E0E0
20400	204020	204040	204060	204080	2040A0	2040C0	2040E0	20600	206020	206040	206060	206080	2060A0	2060C0	2060E0
20800	208020	208040	208060	208080	2080A0	2080C0	2080E0	20A00	20A020	20A040	20A060	20A080	20A0A0	20A0C0	20A0E0
20C00	20C020	20C040	20C060	20C080	20C0A0	20C0C0	20C0E0	20E00	20E020	20E040	20E060	20E080	20E0A0	20E0C0	20E0E0
40800	408020	408040	408060	408080	4080A0	4080C0	4080E0	40E00	40E020	40E040	40E060	40E080	40E0A0	40E0C0	40E0E0
40400	404020	404040	404060	404080	4040A0	4040C0	4040E0	40600	406020	406040	406060	406080	4060A0	4060C0	4060E0
40800	408020	408040	408060	408080	4080A0	4080C0	4080E0	40A00	40A020	40A040	40A060	40A080	40A0A0	40A0C0	40A0E0
40C00	40C020	40C040	40C060	40C080	40C0A0	40C0C0	40C0E0	40E00	40E020	40E040	40E060	40E080	40E0A0	40E0C0	40E0E0
60C00	60C020	60C040	60C060	60C080	60C0A0	60C0C0	60C0E0	60E00	60E020	60E040	60E060	60E080	60E0A0	60E0C0	60E0E0
60400	604020	604040	604060	604080	6040A0	6040C0	6040E0	60600	606020	606040	606060	606080	6060A0	6060C0	6060E0
60800	608020	608040	608060	608080	6080A0	6080C0	6080E0	60A00	60A020	60A040	60A060	60A080	60A0A0	60A0C0	60A0E0
60C00	60C020	60C040	60C060	60C080	60C0A0	60C0C0	60C0E0	60E00	60E020	60E040	60E060	60E080	60E0A0	60E0C0	60E0E0
80C00	80C020	80C040	80C060	80C080	80C0A0	80C0C0	80C0E0	80E00	80E020	80E040	80E060	80E080	80E0A0	80E0C0	80E0E0
80400	804020	804040	804060	804080	8040A0	8040C0	8040E0	80600	806020	806040	806060	806080	8060A0	8060C0	8060E0
80800	808020	808040	808060	808080	8080A0	8080C0	8080E0	80A00	80A020	80A040	80A060	80A080	80A0A0	80A0C0	80A0E0
80C00	80C020	80C040	80C060	80C080	80C0A0	80C0C0	80C0E0	80E00	80E020	80E040	80E060	80E080	80E0A0	80E0C0	80E0E0
A0C00	A0C020	A0C040	A0C060	A0C080	A0C0A0	A0C0C0	A0C0E0	A0E00	A0E020	A0E040	A0E060	A0E080	A0E0A0	A0E0C0	A0E0E0
A0400	A04020	A04040	A04060	A04080	A040A0	A040C0	A040E0	A0600	A06020	A06040	A06060	A06080	A060A0	A060C0	A060E0
A0800	A08020	A08040	A08060	A08080	A080A0	A080C0	A080E0	A0A00	A0A020	A0A040	A0A060	A0A080	A0A0A0	A0A0C0	A0A0E0
A0C00	A0C020	A0C040	A0C060	A0C080	A0C0A0	A0C0C0	A0C0E0	A0E00	A0E020	A0E040	A0E060	A0E080	A0E0A0	A0E0C0	A0E0E0
C0C00	C0C020	C0C040	C0C060	C0C080	C0C0A0	C0C0C0	C0C0E0	C0E00	C0E020	C0E040	C0E060	C0E080	C0E0A0	C0E0C0	C0E0E0
C0400	C04020	C04040	C04060	C04080	C040A0	C040C0	C040E0	C0600	C06020	C06040	C06060	C06080	C060A0	C060C0	C060E0
C0800	C08020	C08040	C08060	C08080	C080A0	C080C0	C080E0	C0A00	C0A020	C0A040	C0A060	C0A080	C0A0A0	C0A0C0	C0A0E0
C0C00	C0C020	C0C040	C0C060	C0C080	C0C0A0	C0C0C0	C0C0E0	C0E00	C0E020	C0E040	C0E060	C0E080	C0E0A0	C0E0C0	C0E0E0
E0C00	E0C020	E0C040	E0C060	E0C080	E0C0A0	E0C0C0	E0C0E0	E0E00	E0E020	E0E040	E0E060	E0E080	E0E0A0	E0E0C0	E0E0E0
E0400	E04020	E04040	E04060	E04080	E040A0	E040C0	E040E0	E0600	E06020	E06040	E06060	E06080	E060A0	E060C0	E060E0
E0800	E08020	E08040	E08060	E08080	E080A0	E080C0	E080E0	E0A00	E0A020	E0A040	E0A060	E0A080	E0A0A0	E0A0C0	E0A0E0
E0C00	E0C020	E0C040	E0C060	E0C080	E0C0A0	E0C0C0	E0C0E0	E0E00	E0E020	E0E040	E0E060	E0E080	E0E0A0	E0E0C0	E0E0E0

邊框

依 **Border**、**BorderWidth** 兩個設定值決定。**Border** 決定上下左右各自是否有框線。**BorderWidth** 決定框線的粗細，粗細無法依據上下左右單獨設定，設定值適用於四周被輸出的框線。

Border 可以用套件中定義的 **enum** 設定。基本的項目就是上**T**下**B**左**L**右**R**。另外各種組合的簡寫定義，可以參考下圖。



文字對齊

欄位中的文字對齊，分別使用`setAlignH`、`setAlignV`定義水平與垂直，對齊效果可參考下圖：

表格合并欄位

PDF : addCell (... , int colspans, ...)

PDF : subTable (...)

Excel : appendCell(ExcelPoint, ExcelPoint, Object, CellFormat)

TODO ADD SAMPLE

TODO ADD SAMPLE

TODO ADD SAMPLE

Temp

2.0.2 移除註記

- ColValue extends HorizontalExpression
 - 對應到其它 column-metadata / 已有替代類別
- DateAddition extends HorizontalExpression {
 - 日期計算，未使用。
- //public class ExcelFunction extends HorizontalExpression {
 - 待重新設計

```
//      //      NumberOP a = new ColNumber(col_1)//  
//      //      .add(col_2)//  
//      //      .add(col_3)//  
//      //      .subtract(col_4);  
//      //      ;  
//      //      ColValue v4 = new ColValue(col_4);  
//      //      ExcelFunction<Object> excelFunction = new ExcelFunction<Ob  
ject>("DIFF", a, v4);
```

- public class ArrayData implements CellDataSource {
 - 當傳入的原始資料為陣列形式.
- 文字切成多行 : LineSplitter // bySeparator // byChars

特殊樣式處理

- 對 PDF 的支援度較高。

CellType

```
public interface Celltype {  
    Celltype NONE = text -> text;  
    String asText(String defaultText); // FOR EXCEL  
}
```

專案若自行定義 Celltype，同時必須實作對應的 CellCreator。

```
public interface CellCreator<T extends Celltype> {  
    PdfPCell create(CHTFontFactory fontFactory, int fontSize, String text, T cellType, Paragraph paragraph);  
}
```

TableiTextDecorator

實作以下 MEHTOD，當每一個單獨表格繪製完成後，可用自訂的TableiTextDecorator加上表格裝飾效果。UDE-Report 也有提供一些常用的裝飾類別。

```
public abstract void drawTableLayout(PDFDocument pdfDocument //  
    , PdfPTable table //  
    , float[][] widths //  
    , float[] heights //  
    , int headerRows //  
    , int rowStart//  
    , PdfContentByte[] canvases);
```

TODO ADD SAMPLE

傳入參數	輸出結果	文字表示	傳入參數	輸出結果	文字表示
地區;人數;年度	人數 地區 年度	地區 人數 年度	地區;人數;null	人數 地區 年度	地區 人數 年度
地區:null;年度	地區 年度	地區 年度	地區:null:null	地區 年度	地區 年度
null;人數;年度	人數 年度	人數 年度	null;人數:null	人數 年度	人數 年度
null:null;年度	年度	年度	null:null:null		

TODO ADD SAMPLE



TODO ADD SAMPLE

P100000_0000			
輸出層輸出層			
輸出層	輸出層	輸出層	0.00
輸出層	輸出層	輸出層	0.00
輸出層	輸出層	輸出層	0.00
輸出層輸出層			
輸出層	輸出層	輸出層	0.00
輸出層	輸出層	輸出層	0.00
輸出層	輸出層	輸出層	0.00
輸出層輸出層			
輸出層	輸出層	輸出層	0.00
輸出層	輸出層	輸出層	0.00
輸出層	輸出層	輸出層	0.00
輸出層輸出層			
輸出層	輸出層	輸出層	0.00
輸出層	輸出層	輸出層	0.00
輸出層	輸出層	輸出層	0.00

TODO ADD SAMPLE

HEADER1	HEADER1	HEADER1
HEADER2	HEADER2	HEADER2
TEST	TEST	TEST
TEST	TEST	TEST
TEST	TEST	TEST
TEST	TEST	TEST
TEST	TEST	TEST
INNER TABLE 整塊內部表格使用同一底色		TEST2
		TEST2
		TEST2
		TEST2
		TEST2
TEST	TEST	TEST
TEST	TEST	TEST
TEST	TEST	TEST
TEST	TEST	TEST
TEST	TEST	TEST
FOOTER	FOOTER	FOOTER

繪圖、定位輸出

PaintTool 用於精準輸出圖形、文字於定位的工具類別。需配合 **Coordinate** 建立 **PaintTool** 物件，預設有**CM_BL**、**CM_TL**、**Default**三個共用實例，其單位分別為公分、公分、**pixel**，座標軸原點分別在左下、左上、左下。

若需設定**PdfContentByte**樣式，請使用**iText**原生API。

線段輸出

- `drawLines(PdfContentByte, Float, Float...)`
- `drawBlock(PdfContentByte, PointF, PointF)`
- `drawBlock(PdfContentByte, PointF, PointF, float offset)`
- `drawBlock(PdfContentByte, SimpleRectangle, float offset)`

文字輸出

- `drawText(PdfContentByte, String, float, float)`
- `drawText(PdfContentByte, String, float, float, float)`
- `drawTextAlignLeft(PdfContentByte, String, float, float)`
- `drawTextAlignLeft(PdfContentByte, String, float, float, float)`
- `drawTextBlock(PdfContentByte, Font, String, float, float, float, float)`
- `drawTextBlock(PdfContentByte, Phrase, float, float, float, float)`

圖片輸出

- `drawImageAlignLeft(PdfContentByte, File, float, float)`
- `drawImageAlignLeft(PdfContentByte, File, float, float, ScaleStrategy)`

表格輸出

- `drawTable(PdfContentByte, TableiText, float, float)`

線段輸出

- `drawLines(PdfContentByte, Float, Float...)`
- `drawBlock(PdfContentByte, PointF, PointF)`
- `drawBlock(PdfContentByte, PointF, PointF, float offset)`
- `drawBlock(PdfContentByte, SimpleRectangle, float offset)`

文字輸出

- `drawText(PdfContentByte, String, float, float)`
- `drawText(PdfContentByte, String, float, float, float)`
- `drawTextAlignLeft(PdfContentByte, String, float, float)`
- `drawTextAlignLeft(PdfContentByte, String, float, float, float)`
- `drawTextBlock(PdfContentByte, Font, String, float, float, float, float)`
- `drawTextBlock(PdfContentByte, Phrase, float, float, float, float)`

圖片輸出

條碼

以 `Coordinate` 建立 `Barcode39Drawer`、`Barcode128Drawer`、`QRCodeDrawer`。再使用 `CodeDrawer.draw(PdfContentByte contentByte, String text, PointF pos, float w, float h, boolean showText)` 輸出

CODE 39

```
protected SimpleRectangle drarBarcode(final PdfContentByte contentByte //
    , final BarcodeSupport barCode, final PointF pos, final int w, final float
    h, final String text,
    final boolean showText) {

    final SimpleRectangle rect = barCode.draw(contentByte, text, pos, w, h, showText);
    contentByte.saveState();
    contentByte.setLineWidth(0.2f);
    contentByte.setColorStroke(Color.lightGray);
    PaintTool.Default.drawBlock(contentByte, rect, 20.0f);
    contentByte.restoreState();
    return rect;
}

@Test
public void test_barCode39() {
    super.createPDF(pdfDocument -> {
        final PdfContentByte contentByte = pdfDocument.getPdfWriter().getDirectContent();
        final Barcode39Drawer barCode39 = new Barcode39Drawer(Coordinate.CM_TL);
        // FIXED SIZE
        drarBarcode(contentByte, barCode39, new PointF(3, 4), 5, 0.8f, "TEST60105", true);
        drarBarcode(contentByte, barCode39, new PointF(10.5f, 4), 5, 0.8f, "TEST60105", false);
        // DEFUALT SIZE
        drarBarcode(contentByte, barCode39, new PointF(8.5f, 8), -1, -1, "TEST60105", false);
    });
}
```

QR-CODE

表格輸出

- `drawTable(PdfContentByte, TableiText, float, float)`

表格描述定義

利用本套件輸出此類表格的設定方式，第一步是先建立 `xxxTableMetadata`，再插入個別欄位定義 (含標頭、資料來源及格式)、群組定義。最後再以 `PDFTableTransfer`、`ExcelTableTransfer` 等表格轉換器，搭配原始資料 List《`JavaBean/Map`》轉換為表格輸出。

UDE-Report 的表格描述基本準則，是 1-1 對應設定直欄標頭與資料來源。如下面建立 `TableMetadata` 的範例，為 `metadata` 增加一個「年度」欄位，其資料值「`new BeanProperty("text1")`」表示由 java bean 的 "text1" 屬性取得內容。

```
final TreeTableMetadata metadata = new TreeTableMetadata ();
metadata.append("年度", new BeanProperty("text1"));
// ...
```

一些新增欄位定義的函式如下，後續章節會再逐一介紹。

```
// 基本新增用法
append(String title)
append(String title, CellDataSource source)
append(String title, float widthWeight)
append(String title, CellDataSource source, float widthWeight)

// 新增後，以 Consumer 操作所加入的欄位描述資訊
append(String title, Consumer<C>)
append(String title, CellDataSource source, float widthWeight, Consumer<C>)

// 以 ColumnDefine 介面新增欄位，通常用於使用 ENUM 定義固定欄位選項。
append(ColumnDefine)

// 加入無標題欄位
append(CellDataSource source, float widthWeight)
```

資料表格

UdeReport 中的 TableMetadata，目前簡單區分為樹狀與巢狀兩種定義方式。

- 樹狀 (TreeTableMetadata)

即一般常見的直欄報表型式，具階層關係的標題列顯示在最上方。除統計值外，每筆資料只佔一列空間；也就是只有標題列葉節點所對應的資料欄位會被顯示。同一列之間的欄位可能有相依關係，如相加、相減。所有資料也可能依其它欄位做分組統計。如下表即是一個典型範例。

區域別	遷出人數			遷入人數			淨增減 人數
	男	女	小計	男	女	小計	
A	10	20	30	5	5	10	20
B	20	30	50	50	50	100	-50
總計	30	50	80	55	55	110	-30

- 巢狀 (NestTableMetadata)

當一筆資料需要用多列空間才足以輸出完整內容時，即可以巢狀表格定義。典型的應用如人員清冊資料，因為許多欄位內容像地址，即需要完整的一列空間才足以顯示。

LIST				
年度		地區	項目	
值1	值2	值3	開始日期	
			結束日期	
100年		臺北市	第一類	
100	90	7,905	2011/02/08	
			2015/08/27	
100年		臺北市	第二類	
100	132	8,683	2011/07/06	
			2015/08/27	

樹狀表格

```
@Test
public void test_basicTable() {
    final TreeTableMetadata metadata = new TreeTableMetadata();
    metadata.getDefaultContentFormat().setAlignV(AlignV.MIDDLE);
    metadata.append("年度", new BeanProperty("text1"), 20);
    metadata.append("地區", new BeanProperty("text2"), 20);
    metadata.append("項目", new BeanProperty("text3"), 20);
    metadata.append("值1", new BeanProperty("value1"), 20);
    metadata.append("值2", new BeanProperty("value2"), 20);
    super.createPDF(this::setPageSizeA5R, pdfDocument -> {
        pdfDocument.writeText("基本表格，標題部分每頁重複顯示");
        final PDFTableTransfer transfer = new PDFTableTransfer(pdfDocument, me
tadata);
        transfer.transTable(SampleVO_OM.testDataset());
    });
    super.createExcel(content -> {
        final ExcelSheet<?> sheet = content.createSheet("A");
        final ExcelTableTransfer transfer = new ExcelTableTransfer(metadata, s
heet);
        transfer.transTable(SampleVO_OM.testDataset());
    });
}
```

上例分別用同一個 `TreeTableMetadata` 轉換表格資料為 PDF / EXCEL 輸出。
可以看到在 PDF 中，每一頁會重複顯示標題部分；而 Excel 則否。

基本表格，標題部分每頁重複顯示

年度	地區	項目	值1	值2
100年	臺北市	第一類	100	99
100年	臺北市	第二類	100	104
100年	臺北市	第三類	100	114
100年	臺北市	第四類	100	38
100年	新北市	第一類	100	151
100年	新北市	第二類	100	48
100年	新北市	第三類	100	18
100年	新北市	第四類	100	23
100年	高雄市	第一類	100	113
100年	高雄市	第二類	100	188
100年	高雄市	第三類	100	23
100年	高雄市	第四類	100	33
101年	臺北市	第一類	101	149
101年	臺北市	第二類	101	84
101年	臺北市	第三類	101	60
101年	臺北市	第四類	101	68
101年	新北市	第一類	101	24
101年	新北市	第二類	101	67
101年	新北市	第三類	101	25

年度	地區	項目	值1	值2
101年	新北市	第四類	101	110
101年	高雄市	第一類	101	173
101年	高雄市	第二類	101	145
101年	高雄市	第三類	101	153
101年	高雄市	第四類	101	55
102年	臺北市	第一類	102	171
102年	臺北市	第二類	102	182
102年	臺北市	第三類	102	169
102年	臺北市	第四類	102	43
102年	新北市	第一類	102	87
102年	新北市	第二類	102	2
102年	新北市	第三類	102	137
102年	新北市	第四類	102	86
102年	高雄市	第一類	102	35
102年	高雄市	第二類	102	121
102年	高雄市	第三類	102	129
102年	高雄市	第四類	102	66
103年	臺北市	第一類	103	10
103年	臺北市	第二類	103	171
103年	臺北市	第三類	103	35

	A	B	C	D	E	F
	年度	地區	項目	值1	值2	
1						
2	100年	臺北市	第一類	100	194	
3	100年	臺北市	第二類	100	111	
4	100年	臺北市	第三類	100	21	
5	100年	臺北市	第四類	100	123	
6	100年	新北市	第一類	100	194	
7	100年	新北市	第二類	100	66	
8	100年	新北市	第三類	100	144	
9	100年	新北市	第四類	100	134	
10	100年	高雄市	第一類	100	36	
11	100年	高雄市	第二類	100	89	
12	100年	高雄市	第三類	100	154	
13	100年	高雄市	第四類	100	3	
14	101年	臺北市	第一類	101	161	
15	101年	臺北市	第二類	101	6	
16	101年	臺北市	第三類	101	165	
17	101年	臺北市	第四類	101	6	
18	101年	新北市	第一類	101	188	
19	101年	新北市	第二類	101	127	

20	101年	新北市	第三編	101	91
21	101年	新北市	第四編	101	151
22	101年	高雄巿	第一編	101	63
23	101年	高雄巿	第二編	101	144
24	101年	高雄巿	第三編	101	27
25	101年	高雄巿	第四編	101	91
26	102年	臺北市	第一編	102	2
27	102年	臺北市	第二編	102	142
28	102年	臺北市	第三編	102	122
29	102年	臺北市	第四編	102	123
30	102年	新北市	第一編	102	176
31	102年	新北市	第二編	102	187
32	102年	新北市	第三編	102	69
33	102年	新北市	第四編	102	162
34	102年	高雄巿	第一編	102	152
35	102年	高雄巿	第二編	102	67
36	102年	高雄巿	第三編	102	9
37	102年	高雄巿	第四編	102	11
38	103年	臺北市	第一編	103	28
39	103年	臺北市	第二編	103	134
40	103年	臺北市	第三編	103	181
41	103年	臺北市	第四編	103	79
42	103年	新北市	第一編	103	162
43	103年	新北市	第二編	103	168

```

@Test
public void test_basicCaption() {
    final TreeTableMetadata metadata = new TreeTableMetadata("標題");
    // ...

```

同一個範例中，若建構 **TreeTableMetadata** 時，加入字串參數，可以指定無框線的標題文字。此標題預設置中、以**14**粗體字型顯示，同樣在 **PDF** 中每一頁重複；在 **Excel** 中只出現一次。若要進一步有更多呈現變化，請參考「額外區塊」一節說明。

標題					
年次	地區	項目	第1	第2	
100年	臺北市	第一類	100	63	
100年	臺北市	第二類	100	12	
100年	臺北市	第三類	100	172	
100年	臺北市	第四類	100	74	
100年	新北市	第一類	100	104	
100年	新北市	第二類	100	9	
100年	新北市	第三類	100	199	
100年	新北市	第四類	100	27	
100年	高雄市	第一類	100	197	
100年	高雄市	第二類	100	63	
100年	高雄市	第三類	100	49	
100年	高雄市	第四類	100	183	
101年	臺北市	第一類	101	112	
101年	臺北市	第二類	101	33	
101年	臺北市	第三類	101	46	
101年	臺北市	第四類	101	60	
101年	新北市	第一類	101	174	

標題					
年次	地區	項目	第1	第2	
101年	新北市	第二類	101	126	
101年	新北市	第三類	101	142	
101年	新北市	第四類	101	73	
101年	高雄市	第一類	101	151	
101年	高雄市	第二類	101	26	
101年	高雄市	第三類	101	157	
101年	高雄市	第四類	101	138	
102年	臺北市	第一類	102	9	
102年	臺北市	第二類	102	176	
102年	臺北市	第三類	102	69	
102年	臺北市	第四類	102	9	
102年	臺北市	第一類	102	148	
102年	新北市	第二類	102	114	
102年	新北市	第三類	102	139	
102年	新北市	第四類	102	139	
102年	高雄市	第一類	102	199	
102年	高雄市	第二類	102	137	
102年	高雄市	第三類	102	167	

標題					
年次	地區	項目	第1	第2	
100年	臺北市	第一類	100	71	
100年	臺北市	第二類	100	166	
100年	臺北市	第三類	100	50	
100年	臺北市	第四類	100	173	
100年	新北市	第一類	100	176	
100年	新北市	第二類	100	45	
100年	新北市	第三類	100	27	
100年	新北市	第四類	100	165	
100年	高雄市	第一類	100	9	
100年	高雄市	第二類	100	63	
100年	高雄市	第三類	100	120	
100年	高雄市	第四類	100	134	
101年	臺北市	第一類	101	169	
101年	臺北市	第二類	101	8	
101年	臺北市	第三類	101	87	
101年	臺北市	第四類	101	103	
101年	新北市	第一類	101	132	
101年	新北市	第二類	101	40	
101年	新北市	第三類	101	120	
101年	新北市	第四類	101	124	
101年	高雄市	第一類	101	195	
101年	高雄市	第二類	101	42	
101年	高雄市	第三類	101	53	
101年	高雄市	第四類	101	162	
102年	臺北市	第一類	102	58	
102年	臺北市	第二類	102	191	
102年	臺北市	第三類	102	101	
102年	臺北市	第四類	102	66	
102年	新北市	第一類	102	70	
102年	新北市	第二類	102	81	
102年	新北市	第三類	102	137	
102年	新北市	第四類	102	67	
102年	高雄市	第一類	102	77	
102年	高雄市	第二類	102	95	
102年	高雄市	第三類	102	110	

樹狀欄位群組設定

在規劃直欄表格時，可能會把相鄰的幾個欄位以群組方式呈現，以利使用者檢視報表。

實作說明

在 UDE-Report 中的用例如下，也就是以 `append(String title, Consumer)` 建立合併欄位("資料內容")後，再在 `Consumer.accept()` 中，對該欄位增加 "值1","值2" 兩個子欄位。

```
metadata.append("資料內容", column -> {  
    column.append("值1", new BeanProperty("value1"));  
    column.append("值2", new BeanProperty("value2"));  
});
```

在 JDK 7 以前的寫法則如下例所示，若只有一兩個欄位有後續資料要設定時還好，但東西一多，會多出不少 **Local variables**，程式的可讀性會略差一些。

```
final TreeColumnMetadata column1 = metadata.append("資料內容");  
column1.append("值1", new BeanProperty("value1"));  
column1.append("值2", new BeanProperty("value2"));  
final TreeColumnMetadata column2 = metadata.append("資料內容");  
column2.append("值3", new BeanProperty("value3"));  
column2.append("值4", new BeanProperty("value4"));
```

產出結果

執行產出如下，沒有子欄位的標題項目，就會變成跨列欄位。

PDF

年度	地區	項目	資料內容	
			值1	值2
101年	新北市	第三類	101	21
101年	新北市	第四類	101	91
101年	高雄市	第一類	101	7
101年	高雄市	第二類	101	140
101年	高雄市	第三類	101	30

Excel

	A	B	C	D	E
1	年度	地區	項目	資料內容	
2				值1	值2
3	100年	臺北市	第一類	100	68
4	100年	臺北市	第二類	100	122
5	100年	臺北市	第三類	100	72

多層次範例

也可以設定3層以上的樹狀關係。

```
metadata.append("所有資料", columnGroup -> {
    columnGroup.append("項目", new BeanProperty("text3"));
    columnGroup.append("資料內容", column -> {
        column.append("值1", new BeanProperty("value1"));
        column.append("值2", new BeanProperty("value2"));
    });
});
```

年度	地區	所有資料		
		項目	資料內容	
			值1	值2
102年	高雄市	第四類	102	129
103年	臺北市	第一類	103	72
103年	臺北市	第二類	103	105

欄位分割

若新增子欄位時，皆不再給定欄位標題，則效果會變成僅標題欄位跨欄合併。

```
metadata.append("年度、地區", areaGroup -> {
    areaGroup.append(new BeanProperty("text1"));
});
```

```

        areaGroup.append(new BeanProperty("text2"));
    });
    metadata.append("資料集", columnGroup -> {
        // 操作邊框定義，以呈現合併效果
        columnGroup.append(new BeanProperty("value1")).getContentFormat().setBorder(Border.NR);
        columnGroup.append(new BeanProperty("value2")).getContentFormat().setBorder(Border.TB);
        columnGroup.append(new BeanProperty("value3")).getContentFormat().setBorder(Border.NL);
    });

```

產出結果

年度、地區		資料集		
100年	臺北市	100	82	5118
100年	臺北市	100	43	2434
100年	臺北市	100	135	625
100年	臺北市	100	117	2086

完整測試案例

```

@Test
public void test_columnGroup() {
    final TreeTableMetadata metadata = new TreeTableMetadata();
    metadata.getDefaultContentFormat().setAlignV(AlignV.MIDDLE);
    metadata.append("年度", new BeanProperty("text1"));
    metadata.append("地區", new BeanProperty("text2"));
    metadata.append("項目", new BeanProperty("text3"));
    metadata.append("資料內容", column -> {
        column.append("值1", new BeanProperty("value1"));
        column.append("值2", new BeanProperty("value2"));
    });
    super.createPDF(this::setPageSizeA5R, pdfDocument -> {
        pdfDocument.writeText("基本表格，標題部分每頁重複顯示");
        final PDFTableTransfer transfer = new PDFTableTransfer(pdfDocument, metadata);
        transfer.transTable(SampleVO_OM.testDataset());
    });
    super.createExcel(excelDocument -> {
        final ExcelSheet<> sheet = excelDocument.createSheet("A");
        final ExcelTableTransfer transfer = new ExcelTableTransfer(metadata, sheet);
        transfer.transTable(SampleVO_OM.testDataset());
    });
}

```


欄位寬度定義

以 **TableMetadata** 定義表格，基本上欄寬皆是以權重方式定義。

append 函式中，基本相關的組合如下，皆是多了一個 **float** 參數以定義寬度權重。

```
// 基本新增用法
append(String title, float widthWeight)
append(String title, CellDataSource source, float widthWeight)
append(CellDataSource source, float widthWeight)

// 新增後，以 Consumer 操作所加入的欄位描述資訊
append(String title, float widthWeight, Consumer<C>)
append(String title, CellDataSource source, float widthWeight, Consumer<C>)
```

實作說明

- 上層節點寬度為下層節點的總合。
- 另外對上層節點設定寬度沒有作用。

```
metadata.append("年度", new BeanProperty("text1"), 10);
metadata.append("地區", new BeanProperty("text2"), 20);
metadata.append("項目", new BeanProperty("text3"), 30);
metadata.append("資料內容", column -> {
    column.append("值1", new BeanProperty("value1"), 10);
    column.append("值2", new BeanProperty("value2"), 10);
});
metadata.append("資料內容", 10, column -> {
    column.append("值3", new BeanProperty("value1"), 10);
    column.append("值4", new BeanProperty("value2"), 10);
});
```

產出結果

可以看到第二組資料內容(值3/值4)的總計寬度，與第一組資料內容(值1/值2)一模一樣；也與「地區」欄位寬度相同。因為它們所佔權重的總合皆為 20。

PDF

年度	地區	項目	資料內容		資料內容	
			值1	值2	值3	值4
101年	新北市	第三類	101	113	101	113
101年	新北市	第四類	101	189	101	189
101年	高雄市	第一類	101	179	101	179
101年	高雄市	第二類	101	61	101	61
101年	高雄市	第三類	101	147	101	147
101年	高雄市	第四類	101	144	101	144
102年	臺北市	第一類	102	1	102	1
102年	臺北市	第二類	102	30	102	30
102年	臺北市	第三類	102	48	102	48
102年	臺北市	第四類	102	32	102	32
102年	新北市	第一類	102	119	102	119
102年	新北市	第二類	102	88	102	88
102年	新北市	第三類	102	137	102	137
102年	新北市	第四類	102	81	102	81
102年	高雄市	第一類	102	198	102	198
102年	高雄市	第二類	102	41	102	41
102年	高雄市	第三類	102	102	102	102
102年	高雄市	第四類	102	129	102	129
103年	臺北市	第一類	103	129	103	129

Excel

年度	地區	項目	資料內容		資料內容	
			值1	值2	值3	值4
101年	新北市	第三類	101	113	101	113
101年	新北市	第四類	101	189	101	189
101年	高雄市	第一類	101	179	101	179
101年	高雄市	第二類	101	61	101	61
101年	高雄市	第三類	101	147	101	147
101年	高雄市	第四類	101	144	101	144
102年	臺北市	第一類	102	1	102	1
102年	臺北市	第二類	102	30	102	30
102年	臺北市	第三類	102	48	102	48
102年	臺北市	第四類	102	32	102	32
102年	新北市	第一類	102	119	102	119
102年	新北市	第二類	102	88	102	88
102年	新北市	第三類	102	137	102	137
102年	新北市	第四類	102	81	102	81
102年	高雄市	第一類	102	198	102	198
102年	高雄市	第二類	102	41	102	41
102年	高雄市	第三類	102	102	102	102
102年	高雄市	第四類	102	129	102	129
103年	臺北市	第一類	103	129	103	129

WidthUnit

在上例中，表格轉換的總計欄寬以頁面寬度的百分比為準(預設為100%)。

但有時會需要以更明確的單位規劃表格，如公分、如 Excel 中的欄寬單位。這時可以使用 `metadata.setWidthUnit(LengthUnit)` 函式設定。若寬度總和小於頁面範圍，可以用 `setHorizontalAlignment` 將預設置中改換為靠左或靠右輸出。

```
// FOR PDF
metadata.setWidthUnit(LengthUnit.MM);
metadata.setHorizontalAlignment(DocumentAlign.LEFT);
// FOR Excel
metadata.setWidthUnit(LengthUnit.ExcelPoint);
});
```

產出結果(WidthUnit)

PDF

年度	地區	項目	資料內容 值1	資料內容 值2	資料內容 值3	資料內容 值4
100年	臺北市	第一類	100	159	100	159
100年	臺北市	第二類	100	75	100	75
100年	臺北市	第三類	100	104	100	104
100年	臺北市	第四類	100	39	100	39
100年	新北市	第一類	100	50	100	50
100年	新北市	第二類	100	57	100	57
100年	新北市	第三類	100	108	100	108
100年	新北市	第四類	100	13	100	13
100年	高雄市	第一類	100	37	100	37
100年	高雄市	第二類	100	26	100	26

年度	地區	項目	資料內容 值1	資料內容 值2	資料內容 值3	資料內容 值4
100年	臺北市	第二類	100	191	100	191
100年	臺北市	第三類	100	117	100	117
100年	臺北市	第四類	100	44	100	44
100年	新北市	第一類	100	46	100	46
100年	新北市	第二類	100	111	100	111
100年	新北市	第三類	100	182	100	182
100年	新北市	第四類	100	135	100	135
100年	高雄市	第一類	100	96	100	96
100年	高雄市	第二類	100	120	100	120

Excel

年度	地區	項目	資料內容 值1	資料內容 值2	資料內容 值3	資料內容 值4
100年	臺北市	第一類	100	102	100	102
100年	臺北市	第二類	100	7	100	7
100年	臺北市	第三類	100	48	100	48
100年	臺北市	第四類	100	14	100	14
100年	新北市	第一類	100	177	100	177

完整測試案例

```
@Test
public void test_widths() {
    final TreeTableMetadata metadata = new TreeTableMetadata();
    metadata.getDefaultContentFormat().setAlignV(AlignV.MIDDLE);
    metadata.append("年度", new BeanProperty("text1"), 10);
    metadata.append("地區", new BeanProperty("text2"), 20);
    metadata.append("項目", new BeanProperty("text3"), 30);
    metadata.append("資料內容", column -> {
        column.append("值1", new BeanProperty("value1"), 10);
        column.append("值2", new BeanProperty("value2"), 10);
    });
    metadata.append("資料內容", 10, column -> {
        column.append("值3", new BeanProperty("value1"), 10);
        column.append("值4", new BeanProperty("value2"), 10);
    });
    super.createPDF(this::setPageSizeA5R, pdfDocument -> {
        pdfDocument.writeText("基本表格，標題部分每頁重複顯示");
        final PDFTableTransfer transfer = new PDFTableTransfer(pdfDocument, metadata);
        transfer.transTable(SampleVO_OM.testDataset());
    });
    super.createExcel(excelDocument -> {
        final ExcelSheet<?> sheet = excelDocument.createSheet("A");
        final ExcelTableTransfer transfer = new ExcelTableTransfer(metadata, sheet);
    });
}
```

```

        transfer.transTable(SampleVO_OM.testDataset());
    });
}

@Test
public void test_widthsUnit() {
    final TreeTableMetadata metadata = new TreeTableMetadata();
    metadata.getDefaultContentFormat().setAlignV(AlignV.MIDDLE);
    metadata.append("年度", new BeanProperty("text1"), 10);
    metadata.append("地區", new BeanProperty("text2"), 20);
    metadata.append("項目", new BeanProperty("text3"), 30);
    metadata.append("資料內容", column -> {
        column.append("值1", new BeanProperty("value1"), 10);
        column.append("值2", new BeanProperty("value2"), 10);
    });
    metadata.append("資料內容", 10, column -> {
        column.append("值3", new BeanProperty("value1"), 10);
        column.append("值4", new BeanProperty("value2"), 10);
    });
    super.createPDF(this::setPageSizeA5R, pdfDocument -> {

        metadata.setWidthUnit(LengthUnit.MM);
        final PDFTableTransfer transfer = new PDFTableTransfer(pdfDocument, me
tadata);

        transfer.transTable(SampleVO_OM.testDataset());

        pdfDocument.newPage();
        metadata.setHorizontalAlignment(DocumentAlign.LEFT);
        transfer.transTable(SampleVO_OM.testDataset());
    });
    super.createExcel(excelDocument -> {
        metadata.setWidthUnit(LengthUnit.ExcelPoint);
        final ExcelSheet<?> sheet = excelDocument.createSheet("A");
        final ExcelTableTransfer transfer = new ExcelTableTransfer(metadata, s
heet);

        transfer.transTable(SampleVO_OM.testDataset());
    });
}

```

欄位插入模式

有時，為配合程式運算邏輯，無法依據由前到後的順序插入欄位。像是輸出於前方的合計欄位、欄位定義在其它形式的資料結構中。

`TreeTableMetadata` 對應的 `TreeColumnMetadata` 提供更多模式的插入欄位方法 (參考JQuery命名)。在「資料來源」一節中，可以看到此種插入模式的應用。

After and Before

```
// !
final TreeColumnMetadata columnValue = metadata.append("值1", new BeanProperty("value1"));
metadata.append("值2", new BeanProperty("value2"));
columnValue.before("值1前面");
columnValue.after("值1後面");

// !
final TreeColumnMetadata columnGroup = metadata.append("資料集");
columnGroup.append("值3", new BeanProperty("value3"));
columnGroup.append("值4", new BeanProperty("value4"));
columnGroup.before("資料集前面");
columnGroup.after("資料集後面");
```

產出結果(After and Before)

年度	地區	值1前面	值1	值1後面	值2	資料集 前面	資料集		資料集 後面	值5
							值3	值4		
100年	臺北市		100		64		8199			
100年	臺北市		100		189		2047			

ColumnGroup : appendAt / prepend

依上層欄位為基準新增：

```
final TreeColumnMetadata columnGroup = metadata.append("資料集");
columnGroup.append("值1", new BeanProperty("value1"));
columnGroup.append("值2", new BeanProperty("value2"));
columnGroup.prepend("資料集第1項");
columnGroup.appendAt("資料集第3項", 2);
metadata.append("值3");
```

產出結果(ColumnGroup)

年度	地區	資料集				值3
		資料集第1項	值1	資料集第3項	值2	
100年	臺北市		100		100	
100年	臺北市		100		90	

完整測試案例

```

@Test
public void test_afterAndBefore() {
    final TreeTableMetadata metadata = new TreeTableMetadata();
    metadata.getDefaultContentFormat().setAlignV(AlignV.MIDDLE);
    metadata.append("年度", new BeanProperty("text1"));
    metadata.append("地區", new BeanProperty("text2"));

    // !
    final TreeColumnMetadata columnValue = metadata.append("值1", new BeanProperty("value1"));
    metadata.append("值2", new BeanProperty("value2"));
    columnValue.before("值1前面");
    columnValue.after("值1後面");

    // !
    final TreeColumnMetadata columnGroup = metadata.append("資料集");
    columnGroup.append("值3", new BeanProperty("value3"));
    columnGroup.append("值4", new BeanProperty("value4"));
    columnGroup.before("資料集前面");
    columnGroup.after("資料集後面");

    metadata.append("值5");

    super.createPDF(this::setPageSizeA5R, pdfDocument -> {
        final PDFTableTransfer transfer = new PDFTableTransfer(pdfDocument, metadata);
        transfer.transTable(SampleVO_OM.testDataset());
    });
    super.createExcel(excelDocument -> {
        final ExcelSheet<?> sheet = excelDocument.createSheet("A");
        final ExcelTableTransfer transfer = new ExcelTableTransfer(metadata, sheet);
        transfer.transTable(SampleVO_OM.testDataset());
    });
}

@Test
public void test_columnGroup() {
    final TreeTableMetadata metadata = new TreeTableMetadata();
    metadata.getDefaultContentFormat().setAlignV(AlignV.MIDDLE);
    metadata.append("年度", new BeanProperty("text1"));

```

```

metadata.append("地區", new BeanProperty("text2"));

// !
final TreeColumnMetadata columnGroup = metadata.append("資料集");
columnGroup.append("值1", new BeanProperty("value1"));
columnGroup.append("值2", new BeanProperty("value2"));
columnGroup.prepend("資料集第1項");
columnGroup.appendAt("資料集第3項", 2);
metadata.append("值3");

super.createPDF(this::setPageSizeA5R, pdfDocument -> {
    final PDFTableTransfer transfer = new PDFTableTransfer(pdfDocument, me
tadata);
    transfer.transTable(SampleVO_OM.testDataset());
});
super.createExcel(excelDocument -> {
    final ExcelSheet<?> sheet = excelDocument.createSheet("A");
    final ExcelTableTransfer transfer = new ExcelTableTransfer(metadata, s
heet);
    transfer.transTable(SampleVO_OM.testDataset());
});
}

```

巢狀表格

```
final NestTableMetadata metadata = new NestTableMetadata();
metadata.append("年度", new BeanProperty("text1"));
metadata.append("地區", new BeanProperty("text2"));
metadata.append("項目", new BeanProperty("text3"));
metadata.subTable(subTable -> {
    subTable.append("值1");
    subTable.append("值2");
    subTable.nextRow();
    subTable.append("值3");
    subTable.append("值4");
    subTable.append("值5");
    subTable.nextRow();
    subTable.append("值A");
    subTable.append("值B");
    subTable.append("值C");
    subTable.append("值D");
});
```

UDE-Report 所定義的巢狀表格，其巢狀新增方式是一塊塊插入子表格的做法。與 `TreeTableMetadata` 相較，產出 PDF 的欄位 LAYOUT 會比較有彈性。如此例中，值1~5/A~C 欄位所佔寬度不同且未對齊；但生成 EXCEL 時，因試算表的天然限制，加上輸出原則以盡量不合併欄位為主，所以生成結果會與 PDF 文件有些許不同。

PDF

年度	地區	項目	值1		值2	
			值3	值4	值5	值6
100年	新北市	第一類	值A	值B	值C	值D
100年	新北市	第二類				
100年	新北市	第三類				
100年	新北市	第四類				
100年	高雄市	第一類				
100年	高雄市	第二類				
100年	高雄市	第三類				

Excel

年度	地區	項目	值1		值2	
			值3	值4	值5	值6
100年	臺北市	第一類	值A	值B	值C	值D
100年	臺北市	第二類				
100年	臺北市	第三類				

資料框線

為使人閱讀清冊報表時，可以較容易區別單筆資料，`NestTableMetadata` 預設會把每一組資料所佔的列，用粗線外框標示。

可使用 `metadata.setBorderWidth(0F)`; 取消此行為。

無框線表格

一般產出無框線表格，也會同時把資料框線設為不顯示。

```
final NestTableMetadata metadata = new NestTableMetadata();
metadata.getDefaultFormat().setBorder(Border.N);
metadata.setBorderWidth(0F);
```

output

無框線表格 加 底線分割

```
final NestTableMetadata metadata = new NestTableMetadata();
metadata.getDefaultFormat().setBorder(Border.N);
metadata.getDefaultHeaderFormat().setBorder(Border.B);
metadata.setBorder(Border.B);
metadata.setBorderWidth(0.25f);
```

output

額外區塊

- **TableHeader** // 只出現一次
- **TableFooter**// 只出現一次
- **TableHeaderRepeat** // 換頁重複出現
- **NoData** // 沒有資料時輸出

資料統計

其它轉換器

Sample_RepeatTable

其它議題

PDF文件安全性設定

TODO : 內文待補

PDFPageEvent

TODO : 内文待補

大型資料表格

TODO : 內文待補