UDE報表輔助元件使用說明

版本：1.0.0

資拓宏宇國際股份有限公司

民國104年8月31日

目　　　錄

[壹、 前言 5](#_Toc428885631)

[一、 特色 5](#_Toc428885632)

[二、 系統需求 5](#_Toc428885633)

[貳、 基本概念 6](#_Toc428885634)

[一、 iText PDF 6](#_Toc428885635)

[(一) 概述 6](#_Toc428885636)

[(二) 頁面結構 6](#_Toc428885637)

[(三) 程式結構 7](#_Toc428885638)

[二、 POI Excel 8](#_Toc428885639)

[(一) 概述 8](#_Toc428885640)

[(二) 程式結構 8](#_Toc428885641)

[參、 設定、開發與測試 9](#_Toc428885642)

[一、 設定檔 9](#_Toc428885643)

[二、 單元測試 10](#_Toc428885644)

[(一) IText 10](#_Toc428885645)

[三、 SAMPLE 11](#_Toc428885646)

[肆、 基本PDF文件操作 12](#_Toc428885647)

[一、 PDFDocument摘要 12](#_Toc428885648)

[(一) 基本文字輸出 12](#_Toc428885649)

[(二) 基本表格輸出 13](#_Toc428885650)

[二、 頁面與邊界設定 17](#_Toc428885651)

[三、 頁首、頁尾 20](#_Toc428885652)

[(一) 基本輸出 20](#_Toc428885653)

[(二) 頁碼 22](#_Toc428885654)

[(三) 分節處理 23](#_Toc428885655)

[(四) 以表格輸出 25](#_Toc428885656)

[四、 浮水印、騎縫章 27](#_Toc428885657)

[(一) 文字浮水印 27](#_Toc428885658)

[(二) 圖片浮水印 32](#_Toc428885659)

[(三) 線段繪製浮水印 33](#_Toc428885660)

[(四) 圖片騎縫章 35](#_Toc428885661)

[(五) 計算騎縫章邊界範圍 37](#_Toc428885662)

[(六) 外部設定檔定義 37](#_Toc428885663)

[五、 文件保全 37](#_Toc428885664)

[伍、 基本Excel 輸出 38](#_Toc428885665)

[一、 ExcelDocument摘要 38](#_Toc428885666)

[二、 格式設定 38](#_Toc428885667)

[三、 合併欄位 38](#_Toc428885668)

[四、 列印設定 38](#_Toc428885669)

[陸、 表格與樣式 39](#_Toc428885670)

[一、 底色 39](#_Toc428885671)

[二、 邊框 40](#_Toc428885672)

[三、 文字、對齊 40](#_Toc428885673)

[四、 跨欄、跨列與子表格(InnerTable) 42](#_Toc428885674)

[五、 特殊欄位輸出 47](#_Toc428885675)

[(一) TrisectionTitle 48](#_Toc428885676)

[(二) ExtraLines 48](#_Toc428885677)

[六、 表格裝飾 51](#_Toc428885678)

[(一) TableHeader 51](#_Toc428885679)

[(二) TableBorder 52](#_Toc428885680)

[(三) RowBackgroundColor 53](#_Toc428885681)

[柒、 通用表格 54](#_Toc428885682)

[一、 概念 54](#_Toc428885683)

[二、 設定方式 54](#_Toc428885684)

[(一) 直列表格 54](#_Toc428885685)

[(二) 清冊表單 61](#_Toc428885686)

[三、 特殊輸出 66](#_Toc428885687)

[(一) 多欄表格 66](#_Toc428885688)

[(二) 欄位跨頁表格 68](#_Toc428885689)

[(三) 大量表格 68](#_Toc428885690)

[捌、 其它議題 69](#_Toc428885691)

[一、 PaintTool 69](#_Toc428885692)

[(一) 線段輸出 69](#_Toc428885693)

[(二) 文字輸出 69](#_Toc428885694)

[(三) 圖片輸出 69](#_Toc428885695)

[(四) 表格輸出 69](#_Toc428885696)

[二、 條碼輸出 69](#_Toc428885697)

[三、 PageEvent 70](#_Toc428885698)

|  |  |  |
| --- | --- | --- |
| 版本編號 | 增(修) 訂日期 | 主要修訂摘要 |
| 1.0.0 | 2015/08/31 | 初版 |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

# 前言

UDE報表輔助元件是基於 iText 及 Apache POI 開發的報表產製輔助套件。其中大幅簡化iText及Apache POI產製PDF及Excel的文件的程式語法。並針對通用性的表格，提供描述資料定義，可使用同一組描述資料與資料集，產出不同格式的輸出結果，減少重複程式開發。

## 特色

* PDF 報表產製、條碼、浮水印及檔案上鎖等功能。
* 多樣 PDF 頁首頁尾支援。
* 精準控制 PDF 輸出。
* 支援中文 unicode 2字面、15字面。
* 保留原生 api 使用彈性。

## 系統需求

* JDK 1.7
* iText 2.1.7
* POI
* 全字庫中文字型檔(選配)

# 基本概念

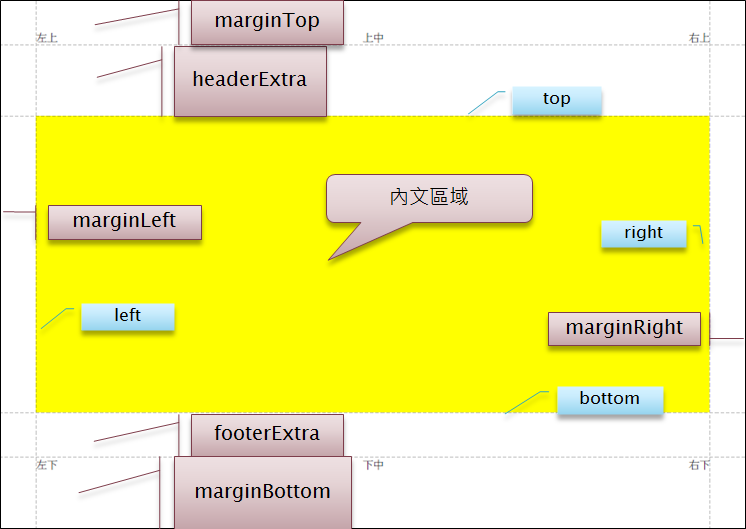
## iText PDF

### 概述

一份完整PDF文件，包含幾個組成部份：封面、目錄、內頁。以頁面結構來看，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。本套件除了產製只有內頁的報表文件外，也提供產製完整文件的相關能力。

### 頁面結構

要精確的呈現需求訪談時所得到的表樣(或反過來引導客戶決定制式表樣)，首先必需對頁面的組成、各元素輸出的基準位置有一致的認知。



上圖是一頁橫式的A4頁面，使用iText 原生Document API，只會得到圖中藍底的4個值(top、bottom、left、rigth)、由這4個定位值所框起來的黃色區域，就是文件內容的輸出範圍。傳統方法在輸出頁首、頁尾時，也會利用上述4個值，再做加減運算，通常會加上一定的留白距離(圖中headerExtra/footerExtra)以保持美觀或供輸出其它內容運用。

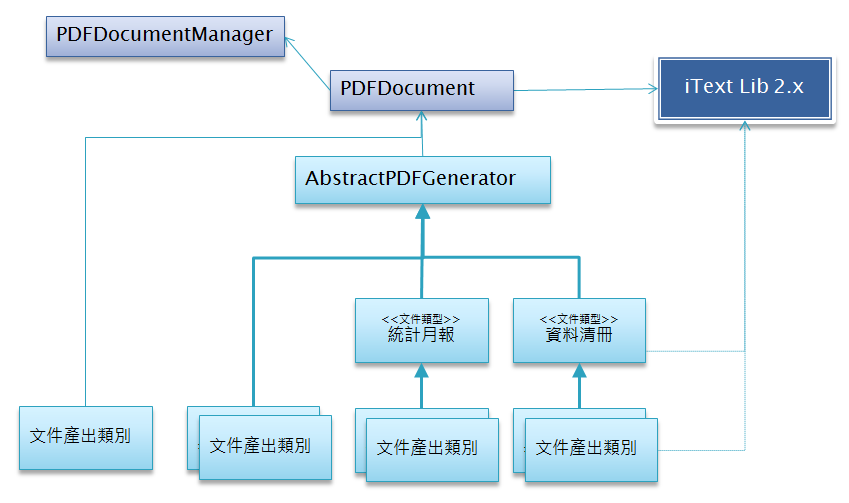
這些數值在iText 中預設原點為左下角，單位為pixel，由於iText PDF 預設為 72 dpi，所以輸出時1公分有 72/ 2.54=28.346 pixel。本套件另外提供轉換機制，方便人工調整輸出樣式、決定文字、圖片、表格輸出位置。可使用公分為單位，並選擇原點在左下或左上。



表格的標題與文件的頁首、頁尾是獨立的兩個概念。文件的頁首是每一頁都會重複，且位置固定；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。

### 程式結構

使用iText產製文件的基本步驟是建立Dodument、PdfWriter物件，並使用相關methods 進行內容輸出。本套件把開關文件的相關流程做了完整的包裝。



相關類別結構如上圖所示：其中PDFDocument 是主要與 iText2 Library相關的基本類別，其負責文件建立、關閉、儲存及取得iText原生類別。PDFDocumentManager 是PDFDocument的Factory類別，負責讀入基本環境設定檔。

AbstractPDFGenerator定義了完整產出文件至檔案的Template Method，開發者只需要在子類別實作generateContent (PDFDocument) 以產出文件實際內容。要注意這個類別非Thread-safe，產出個別 PDF 文件時，應重新建立實作類別Instance。

因應各別系統需求，建議先實作一層共用類別。即圖中標示<<文件類型>>的類別，如統計月報／資料清冊，可依不同種類文件，統一設定頁首頁尾格式或其它輸出風格。

至於實際的文件產出類別，原則上依報表設計文件，一種一類，命名原則各系統自行統一即可，建議規範是放在結尾為.report　的package下，文件類別以Report結尾，如單獨抽出製表部分邏輯，則以Table結尾。

製表時，把相關資料物件先準備完成，在製表前完成設值，並呼叫PDFInfo generate　(PDFDocumentManager pdfDocumentManager, File outputFile)。但資料量過大（如幾十ＭＢ），可能會造成outOfMemory的大型表格，需要用其它方式處理，見特定議題章節討論。

## POI Excel

### 概述

### 程式結構

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。

# 設定、開發與測試

## 設定檔

套件所需要的設定檔為 .properties 格式，預設使用的 itext-config-default.properties 在resources中，內容如下：

|  |
| --- |
| # 預設字型  default.font=CNS11643.SUNG  default.font.size=12  # 預設邊界  default.document.marginLeft=30  default.document.marginRight=30  default.document.marginBottom=30  default.document.marginTop=30  # 預設文件保全  default.encryption.enable=false  default.encryption.owner.password=  default.encryption.user.password=  # 浮水印設定檔位置  watermarks.config.path=${share.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 = ${global.share.path}/fonts/kaiu.ttf  #  # http://www.cns11643.gov.tw/AIDB/cns\_authorization\_apply.do  # 逕行至「政府資料開放平臺」http://data.gov.tw/node/5961下載  #  font.CNS11643.SUNG.0=${global.share.path}/fonts/TW-Sung-98\_1.ttf  font.CNS11643.SUNG.2=${global.share.path}/fonts/TW-Sung-Ext-B-98\_1.ttf  font.CNS11643.SUNG.F=${global.share.path}/fonts/TW-Sung-Plus-98\_1.ttf  font.CNS11643.KAI.0=${global.share.path}/fonts/TW-Kai-98\_1.ttf  font.CNS11643.KAI.2=${global.share.path}/fonts/TW-Kai-Ext-B-98\_1.ttf  font.CNS11643.KAI.F=${global.share.path}/fonts/TW-Kai-Plus-98\_1.ttf |

先就字型部分定義說明：default.font標示文件產出時預設使用的字型代碼，default.font.size標示預設字型大小。

各字型代碼實際使用的字型檔案應以 font.{字型代碼}.{字面} 進行定義，如 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。如相同的字面設定有多組，則套件會逐一測試各檔案路徑是否存在可用字型檔。

若使用完整UDE套件及相關設定方式，設定路徑中可使用${global.share.path}等環境變數，讀入時會自動代換。

各專案若要自行定義此設定檔，可在 Spring Context 設定定義

PDFDocumentManager時，加入建構參數指定ResourcePath，如：

|  |
| --- |
| <bean id="udeFileSystem"  class="com.iisigroup.ude.report.itext2.PDFDocumentManager">  <constructor-arg index="0"  value="/home/project/itext/itext-config.properties" />  </bean> |

## 單元測試

### IText

單元測試類別繼承 AbstractITextTest，建構時可以使用ITextTestConfig定義測試設定值，如檔案輸出位置等。

測試方法中，直接呼叫 super.doTest(AbstractPDFGenerator generator) 即可，AbstractITextTest會以測試類別、方法為名，輸出PDF至指定輸出路徑。

以下是ITextTestConfig中的可指定項目：

* configPath : iText設定檔路徑，預設為 classpath:itext-config-default.properties。
* globalSharePath : 用於取代設定檔中 ${globalSharePath} 的值。預設為.
* 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。
* keepOutputPDF :執行完成後，是否保留PDF檔，預設為TRUE。
* 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。

## SAMPLE

見ude-report-itext-sample專案，其中為簡化 Sample 撰寫，且相容於 JDK 1.7，所以相關範例使用 Groovy 撰寫。但除了Closure 外，範例程式盡量保持語法為 JAVA 格式，而非 Groovy 簡化版本。

另外，所有範例程式輸出，都有設定showMarginBorder為TRUE，所以文件中的PDF產出圖片，都會見到邊界虛線。

# 基本PDF文件操作

## PDFDocument摘要

當使用AbstractPDFGenerator作為文件基底類別時，開發人員無需呼叫PDFDocument物件負責文件建立、關閉、儲存的相關Method。一般會使用的功能，如取得iText原生類別、版面、頁碼、浮水印控制、文字輸出、換頁控制等等。將於後文一一介紹。

### 基本文字輸出

writeText(…) 提供文字輸出基本功能，可設定字型大小及對齊方式。

|  |
| --- |
| Sample\_001\_Basic\_Text.test\_NormalText()  String text = "中文，第二字面：「" + "𠀝" + "」(下/上)";  pdfDocument.writeText(text);  pdfDocument.writeText(text + "Size:10", 10);  pdfDocument.writeText(text + "Size:18", 18);  pdfDocument.writeText(text, 10, DocumentAlign.RIGHT);  pdfDocument.writeText(text, 10, DocumentAlign.CENTER); |

如欲修改文件預設字型，可使用setFontInfo(…)調整。首先要取得指定字體的CHTFontFactory，再用它建立想要產出的字型資訊（FontInfo），可設定大小、Style、顏色、底色。

|  |
| --- |
| final CHTFontFactory kaiFactory  = CHTFontFactories.INSTANCE.getFactory(WindowsFont.KAI);  pdfDocument.setFontInfo(kaiFactory.createFontInfo(10, Color.BLUE));  pdfDocument.writeText("轉成標楷體，無第二字面字體");  pdfDocument.writeText(text); pdfDocument.writeText(text + "Size:10", 10);  pdfDocument.writeText(text + "Size:18", 18);  pdfDocument.writeText(text, 10, DocumentAlign.RIGHT);  pdfDocument.writeText(text, 10, DocumentAlign.CENTER); |

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。

其它特殊格式的文字，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，也可用於設定iText原生Paragraph物件中的各項設定值，如對齊、縮排等等…。

|  |
| --- |
| Sample\_001\_Basic\_Text.test\_Builder()  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("𠀝", redFont) //  .addText("(下/上)", kaiFont);  builder.appendMe(); |

### 基本表格輸出

本套件主要使用iText中的PdfPTable做為底層表格輸出元件。但使用時，應使用com.iisigroup.ude.report.itext2.table.TableiText。

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於製表常用的諸多特性都有進行包裝，後續會在表格輸出的相關章節中有詳細說明。但本套件另有提供通用表格特性，可使用通用表格的時候，還是建議盡量使用通用表格，以簡化程式開發。TableiText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。

要建立TableiText，應使用PDFDocument 的相關函式，當所有欄位加入完成後，呼叫它的appendMe()進行輸出。如欲輸出於頁面上的指定位置，則是使用PaintTool.drawTable(…) method。

#### createTable(float widthPercentage, int numColumns)

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

|  |
| --- |
| Sample\_002\_Basic\_Table. test\_Monospaced ()  pdfDocument.writeText ("等寬表格，指定表格寬度佔頁面比例80")  // 由outputTable處理表格內容資料…及輸出  outputTable(pdfDocument.createTable(80, 5)); |

#### 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。

|  |
| --- |
| Sample\_002\_Basic\_Table. test\_Widths ()  pdfDocument.writeText ("指定欄寬比例，及表格寬度佔頁面比例")  float[] widths = [1, 2, 3, 4, 5];  outputTable(pdfDocument.createTable(90,widths)); |

#### createTable(Coordinate c, float[] realWidths)

指定度量單位c，如Coordinate.CM\_BL即為公分，並直接指定各欄的寬度realWidths。

當寬度超過頁面大小時，則自動以滿版顯示。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。

#### createTable(Coordinate c, float totalWidth, float[] widths)

指定度量單位c、表格總寬度totalWidth，並指定欄寬比例widths。

|  |
| --- |
| Sample\_002\_Basic\_Table. test\_WidthInCm)  pdfDocument.writeText ("指定欄寬比例，及表格寬度(公分)" );  float[] widths = [1, 2, 3, 4, 5];  pdfDocument.writeText ("$CHINESE\_BLOCK 自動加總為15公分：");  outputTable(pdfDocument.createTable(Coordinate.CM\_BL,widths)); |
| pdfDocument.writeText ("$CHINESE\_BLOCK 8公分：");  outputTable(pdfDocument.createTable(Coordinate.CM\_BL,8,widths)) |

|  |
| --- |
| pdfDocument.writeText ("$CHINESE\_BLOCK 500公分  (超過頁寬、以100%計算)：");  outputTable(pdfDocument.createTable(Coordinate.CM\_BL,500,widths));; |

## 頁面與邊界設定

以PDFDocumeneManager產生PDFDocument時，即可設定頁面大小，使用AbstractPDFGenerator，則可以override setup() method，在其中呼叫 setPageSize() 進行定義。

頁面邊界則是經由 LayoutInfo 物件設定，同樣可以在產生PDFDocument時傳入設定或另行設值，範例中可以經由測試用的邊界虛線看出設定效果。

|  |
| --- |
| Sample\_010\_Pages\_Margin  final float marginLeft = Coordinate.CM\_BL.trans(2.5f); // 以公分為單位  final float marginRight = 40f; // 以 pixel 為單位  final float marginTop = 50f;  final float marginBottom = 80f;  final LayoutInfo layoutInfo = new LayoutInfo(marginLeft, marginRight, marginTop, marginBottom);  layoutInfo.setFooterExtra(Coordinate.CM\_BL.trans(1f));  layoutInfo.setHeaderExtra(80);  this.setPageSize(PageSize.A4.rotate());  this.setLayoutInfo(layoutInfo); |

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(“”)。

|  |
| --- |
| Sample\_010\_Pages\_BlankPage  pdfDocument.writeText("P1. 連續兩次newPage不會有空白頁"); pdfDocument.newPage();  pdfDocument.newPage();  pdfDocument.writeText(  "P2. 要新增空白頁，應於newPage間加入頁面元素");  pdfDocument.newPage();  pdfDocument.writeText("");  pdfDocument.newPage();  pdfDocument.writeText("P4. 我是第四頁"); |

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。當PageChanged為真，則換頁事件有被觸發。當NewPageBegin為真，表示還沒有任何內容輸出到新的頁面。

|  |
| --- |
| Sample\_010\_Pages\_PageChangeCheck  for(int i=0;i<200;i++){  pdfDocument.writeText( i + "-1換頁測試\n"  + i + "-2換頁測試\n"  + i + "-3換頁測試\n");  if (pdfDocument.isPageChanged()){  if (pdfDocument.isNewPageBegin()){  pdfDocument.writeText(  "前項輸出導致換頁，新頁面還沒有內容輸出");  } else {  pdfDocument.writeText(  "前項輸出導致換頁，新頁面已有內容輸出");  }  }  } |

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，頁面變動會在下一頁才生效。但是邊界變動可能會導致部分頁首頁尾輸出不如預期，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。

但是一般情況下，不建議做出頁面大小變動，對於使用者要列印輸出或其它處理可能會有困擾。

|  |
| --- |
| Sample\_010\_ChangePageSize  // 原訂大小為 PageSize.A4  // 範例中依序將頁面變動為PageSize.A4, PageSize.A5, PageSize.A6, PageSize.A5, PageSize.A4.rotate() 輸出 |

## 頁首、頁尾

### 基本輸出

關於頁首、頁尾控制，同樣是經由 LayoutInfo 設定。

#### setHeader(final HeaderPosition position, String text, int fontSize)

於頁首、頁尾輸出固定文字text並指定字體大小。輸出位置由position指定，依左中右／上下組合，共有六個位置。文字輸出於左邊時置左；右邊時置右；中間時置中。

文字中若有換行，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。但若預留空間不足，則輸出內容會超出頁首、尾範圍。

另外，這個呼叫方法不支援非0字面輸出。

下例以全黑的方框字元，呈現輸出於不同位置的對齊結果。

|  |
| --- |
| Sample\_011\_Headers\_BASIC\_BASELINE  final LayoutInfo layoutInfo = getPdfManager().createLayoutInfo();  final String text = CHINESE\_BLOCK + "中文" + CHINESE\_BLOCK  + "\n" + CHINESE\_BLOCK + "第二行" + CHINESE\_BLOCK;  layoutInfo.setHeader(HeaderPosition.LeftHeader, text, 18);  layoutInfo.setHeader(HeaderPosition.RightHeader, text, 10);  layoutInfo.setHeader(HeaderPosition.CenterHeader, text, 10);  layoutInfo.setHeader(HeaderPosition.LeftFooter, text, 18);  layoutInfo.setHeader(HeaderPosition.RightFooter, text, 10);  layoutInfo.setHeader(HeaderPosition.CenterFooter, text, 10);  this.setLayoutInfo(layoutInfo); |

#### setHeader(final HeaderPosition position, final HeaderStyle header)

利用不同的 HeaderStyle，設定其餘特殊的頁首、頁尾輸出，包括後續小節提及的頁碼、表格輸出。

頁首、頁尾若要用不同的字型、顏色輸出固定文字時，可使用StringHeader進行定義。字體定義如「基本文字輸出」說明，同樣使用FontInfo設定。

|  |
| --- |
| Sample\_011\_Headers\_BASIC  final CHTFontFactory risFontFactory = CHTFontFactories.INSTANCE.getFactory(CNS11643.SUNG);  final FontInfo textBgGray10 = risFontFactory.  createFontInfo(10,FontStyle.NORMAL, Color.BLACK);  final FontInfo textBgGray12 = risFontFactory.  createFontInfo(14, FontStyle.BOLD, Color.BLACK, Color.LIGHT\_GRAY);  final StringHeader strHeader1 = new StringHeader(//  new SubPhrase(textBgGray10, "多種樣式、特殊字、換行範例\n"),//  new SubPhrase(textBgGray10, "|𠀡|𠀖|"),//  new SubPhrase(textBgGray12, "|𠀡|𠀖|" + "第２字面中文\n"),//  new SubPhrase(textBgGray10, "|𠀡|𠀖|") //  );  layoutInfo.setHeader(HeaderPosition.CenterHeader, strHeader1);  layoutInfo.setHeader(HeaderPosition.CenterFooter, strHeader1);  this.setLayoutInfo(layoutInfo); |

### 頁碼

基本的頁碼輸出，可使用PageHeaderEN / PageHeaderCH 這兩個AbatractPageHeader的實作類別。PageHeaderEN為英文頁碼預先定義，PageHeaderCH中文頁碼預先定義。

建構子的第一個參數PagePattern，決定輸出頁碼的內容含括哪些數值，可用定義如下。

* PAGE 文件頁次
* TOTAL\_PAGES 文件頁數
* BOTH 文件頁次＋文件頁數
* SECTION 目前節次
* PAGE\_IN\_SECTION 目前節內頁次
* SECTION\_AND\_PAGE 目前節次＋目前節內頁次
* SECTION\_PAGES 各節總頁數

建構子的第二個參數就是字型大小。

|  |
| --- |
| Sample\_011\_Headers\_PAGE\_Basic  final LayoutInfo layoutInfo = new LayoutInfo();  layoutInfo.setHeader(LeftHeader, new PageHeaderEN(PAGE, 14));  layoutInfo.setHeader(CenterHeader, new PageHeaderEN(BOTH, 14));  layoutInfo.setHeader(RightHeader, new PageHeaderEN(TOTAL\_PAGES, 14));  layoutInfo.setHeader(LeftFooter, new PageHeaderCH(PAGE, 14));  layoutInfo.setHeader(CenterFooter, new PageHeaderCH(BOTH, 14));  layoutInfo.setHeader(RightFooter, new PageHeaderCH(TOTAL\_PAGES, 14)); |

如果想要改變頁碼顯示的PATTERN，可以設定perfix/conjunction/suffix以變動前後綴與連結文字。當頁碼PATTERN定義對應 1項數值時，輸出為Prefix + $1 + suffix。頁碼PATTERN定義對應 2項數值時，輸出為Prefix + $1 + conjunction+ $2 + suffix。

輸出頁碼所用的數值格式也可以調整，例如有些表樣希望頁碼補0。這裡的數值格式設定使用java String.format 格式，預設為%d。各設定方法的適用項目如下：

* setSectionFormat 目前節次
* setPageFormat 文件頁次、目前節內頁次
* setTotalPagesFormat 文件頁數、各節總頁數

|  |
| --- |
| Sample\_011\_Headers\_PAGE\_PREFIX\_SUFFIX  final LayoutInfo layout = new LayoutInfo();  layout.setHeader(LeftHeader, new PageHeaderEN(PAGE, 14)  .setPrefix("頁次 Page："));  layout.setHeader(CenterHeader, new PageHeaderEN(BOTH, 14)  .setPrefix(" - ").setConjunction("／").setSuffix(" - "));  layout.setHeader(RightHeader, new PageHeaderEN(TOTAL\_PAGES, 14)  .setPrefix("總頁數 Total Pages："));  layout.setHeader(LeftFooter, new PageHeaderCH(PAGE, 14)  .setPrefix("頁次：第").setSuffix("頁"))  .setPageFormat("%04d");  layout.setHeader(CenterFooter, new PageHeaderCH(BOTH, 14)  .setPrefix("頁次：第").setConjunction("頁／共有"))  .setTotalPagesFormat("%4d").setPageFormat("%04d");  layout.setHeader(RightFooter, new PageHeaderCH(TOTAL\_PAGES, 14)  .setPrefix("總頁數：共").setSuffix("頁"))  .setTotalPagesFormat("%4d"); |

### 分節處理

下例展示如何輸出含封面頁(不列計頁碼)的PDF文件。

在各小節開頭，呼叫newSection(); 切換到下一節。

|  |
| --- |
| Sample\_012\_Cover  // 輸出封面頁(無頁碼)  pdfDocument.setSkipPager(true);  pdfDocument.writeText("封面....");  //  for (int s = 1; s <= 5; s++) {  pdfDocument.newSection();  pdfDocument.setSkipPager(true);  int section = pdfDocument.getSection();  pdfDocument.writeText("第" + section+ "小節封面....");  pdfDocument.newPage();  pdfDocument.setSkipPager(false);  for (int i = 1; i <= 0 + s; i++) {  pdfDocument.newPage();  pdfDocument.writeText("TEST PAGE: " + s + "-" + i);  }  } |

### 以表格輸出

用表格輸出頁首頁尾，必須實作AbstractTableHeader，在TableiText createTable(PDFDocument pdfDoc, PageCounter counter) 中實作要輸出的表格物件。其中第二個傳入參數會帶入目前的所有頁次資訊。

getTableHeight、getTableWidth 的預設實作，會回傳開發者定義的表格實際大小，表格輸出的定位點將利用這兩個值計算與邊界之間的相對位置。如果有特別輸出需求，可以利用覆寫實作的方式進行調整。

因為製作各頁內容的當下，不會知道文件會有多少頁，當然其中的總頁次資訊也不會是最後的結果。所以要輸出總頁次，必須要用PdfTemplate處理。AbstractTableHeader.getTemplate() 可取得最後用於輸出總頁次的PdfTemplate，AbstractTableHeader. createTemplateCell() 可以利用產出的PdfTemplate建立表格欄位。

|  |
| --- |
| Sample\_013\_TableHeaders\_PAGE\_TABLE\_Normal  //  layoutInfo.setHeader(LeftHeader, new TABLE\_HEADER(12, 1)); layoutInfo.setHeader(LeftFooter, new TABLE\_HEADER(12, 1, Coordinate.CM\_BL.trans(8)));");  layoutInfo.setHeader(RightFooter, new TABLE\_HEADER(12, 1, Coordinate.CM\_BL.trans(6)));"); |
| InnerClass : TABLE\_HEADER  protected TableiText createTable( … ) {  final float[] widths = { 2.5f, 2f };  final String pageNoStr = String.format("%04d", counter.getPageNow());  final TableiText table = pdfDoc.createTable(Coordinate.CM\_BL, widths);  final CellFormat cf1 = super.createCellFormat() //  .setPadding(this.padding)//  .setAlignH(CellFormat.AlignH.RIGHT)//  .setBorder(Border.NR);  final CellFormat cf2 = super.createCellFormat()//  .setPadding(this.padding)//  .setAlignH(CellFormat.AlignH.LEFT) //  .setBorder(Border.NL);  table.addCell("製表日期：", cf1);  table.addCell("104/01/02", cf2);  table.addCell("頁　　次：", cf1);  table.addCell(pageNoStr, cf2);  table.addCell("總 頁 數：", cf1);  final int width = this.templateFontSize \* 5;  int section = counter.getSection();  final PdfTemplate template =  super.getTemplate(pdfDoc.getPdfWriter(), section, width);  table.addCell(super.createTemplateCell(table, cf2, template));  return table;  }  @Override  protected float getTableWidth(SimpleRectangle headerRect) {  if (this.fixedWidth <= 0) {  return super.getTableWidth(headerRect);  } else {  return this.fixedWidth;  }  } |

## 浮水印、騎縫章

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。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。

|  |
| --- |
| Sample\_014\_Watermark\_TextWaterMark\_BASIC  final TextWatermarkProperties markProp =  new TextWatermarkProperties("測試測試「" + "𠀝" + "」");  markProp.setFontType(CNS11643.KAI);  markProp.setFontSize(48);  super.setMarkInfos(new PropertiesMarkInfo(markProp)); |

旋轉角度的定義值為 0~360，但有兩個特殊定義：-1、-2。-1即為預設的左下-右上對角，-2為左上-右下對角輸出。

|  |
| --- |
| Sample\_014\_Watermark\_TextWaterMark\_Backslash  final TextWatermarkProperties markProp =  new TextWatermarkProperties("測試測試「" + "𠀝" + "」");  markProp.setFontType(CNS11643.KAI);  markProp.setFontSize(48);  markProp.setRotate(-2);  super.setMarkInfos(new PropertiesMarkInfo(markProp)); |

設定浮水印時的markInfos如同前述為List，所以多個浮水印設定是可以同時生效的，下例同時示範TextWatermarkProperties可用的其它設定值。

|  |
| --- |
| Sample\_014\_Watermark\_TextWaterMark\_Multiple  final TextWatermarkProperties markProp1 =  new TextWatermarkProperties("測試測試「" + "𠀝" + "」");  markProp1.setFontType(CNS11643.KAI);  markProp1.setFontSize(40);  markProp1.setRotate(30);  markProp1.setColor(Color.blue);  markProp1.setBold(true);  markProp1.setItalic(true);  markProp1.setUnderline(true);  markProp1.setStrikethru(true);  markProp1.setX(-150);  markProp1.setY(0);  final TextWatermarkProperties markProp2 = SerializationUtils.clone(markProp1);  markProp2.setRotate(60);  markProp2.setFontSize(30);  markProp2.setX(250);  markProp2.setY(0);  markProp2.setAlign(DocumentAlign.LEFT);  final TextWatermarkProperties markProp3 = SerializationUtils.clone(markProp2);  markProp3.setUnderline(false);  markProp3.setStrikethru(false);  markProp3.setAlign(DocumentAlign.RIGHT);  super.setMarkInfos(new PropertiesMarkInfo(markProp1)  , new PropertiesMarkInfo(markProp2)  , new PropertiesMarkInfo(markProp3)); |

若浮水印需要輸出空心文字，可使用setHollow，但此時會使粗體、底線、刪除線設定無效化。

|  |
| --- |
| Sample\_014\_Watermark\_TextWaterMark\_Hellow  //  final TextWatermarkProperties markProp  = new TextWatermarkProperties("測試「" + "𠀝" + "」");  markProp.setFontType(CNS11643.KAI);  markProp.setColor(Color.GREEN);  markProp.setFontSize(80);  markProp.setHollow(true)  super.setMarkInfos(new PropertiesMarkInfo(markProp)); |

另外一組特殊的浮水印設定值是SplitX與SplitY，這兩個值都預設為1。若SplitX設為N，則文件視為水平切割；同理SplitY設為M，則文件視為垂直切割，共輸出N \* M份浮水印文字。

若SplitX設為0，文字錨點對齊左邊界；若SplitX設為0，文字錨點對齊下邊界；當皆設為0時，文字錨點會在左下角。

|  |  |
| --- | --- |
| Sample\_014\_Watermark\_TextWaterMark\_Split\_xxx  doSplitTest(Rectangle pageSize, int sx, int sy, DocumentAlign align) {  super.setPageSize(pageSize);  final TextWatermarkProperties markProp =  new TextWatermarkProperties("測試「" + "𠀝" + "」");  markProp.setSplitX(sx);  markProp.setSplitY(sy);  markProp.setFontSize(18);  markProp.setColor(Color.BLUE);  if (align!=null){markProp.setAlign(align);}  super.setMarkInfos(new PropertiesMarkInfo(markProp))  } | |
| 0X0 | 2x2 |
|  |  |
| 3x0 | 0x3 |
|  |  |

### 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。若打包時會把圖檔資源放到classpath中，則可以用classpath:${resource-path}格式指定。

|  |
| --- |
| Sample\_014\_Watermark\_TextWaterMark\_Image  final ImageWatermarkProperties markProp =  new ImageWatermarkProperties("classpath:RCSS00001.png");  super.setMarkInfos(new PropertiesMarkInfo(markProp)); |

圖片浮水印可以設定旋轉、縮放、濃淡屬性。

圖片浮水印暫時不支援splitX/Y 頁面切割複製功能。

### 線段繪製浮水印

支援以簡單線條繪製浮水印，使用LinesWatermarkProperties。

在線段資料LineInfo中，設定與浮水印錨點的相對位置。

下例將文件4等分，並在各自中心畫出一個十字。

|  |
| --- |
| Sample\_015\_Watermark\_Lines\_Cross  List<LineInfo> lines = [];  lines.add(new LineInfo(new PointF(0, 20), new PointF(0, -20)));  lines.add(new LineInfo(new PointF(20, 0), new PointF(-20, 0)));  final LinesWatermarkProperties markProp = new LinesWatermarkProperties(lines);  markProp.setSplitX(2)  markProp.setSplitY(2)  super.setMarkInfos(new PropertiesMarkInfo(markProp)); |

另一個組合範例輸出文件中心的「機關印信字樣」

|  |
| --- |
| Sample\_015\_Watermark\_Lines\_MsSeam  int r = 80;  List<LineInfo> lines = [];  LineInfo lineDiamond = new LineInfo(  new PointF(r, 0) //  , new PointF(0, r) //  , new PointF(-r, 0) //  , new PointF(0, -r) //  , new PointF(r, 0) //  )  lineDiamond.setColor(Color.LIGHT\_GRAY);  lines.add(lineDiamond);  final LinesWatermarkProperties markProp = new LinesWatermarkProperties(lines);  final int dw = 14; // 位移  final int ch = 14; // 字高  final TextWatermarkProperties p1 = create( "機", +dw, +ch);  final TextWatermarkProperties p2 = create( "關", +dw, -ch);  final TextWatermarkProperties p3 = create( "印", -dw, +ch);  final TextWatermarkProperties p4 = create( "信", -dw, -ch);  super.setMarkInfos(PropertiesMarkInfo.build(Arrays.asList(markProp, p1, p2, p3, p4))); |

### 圖片騎縫章

宣告SealstampProperties時，必須傳入圖檔路徑，與圖片浮水印相同，若打包時會把圖檔資源放到classpath中，也可以用classpath:${resource-path}格式指定。

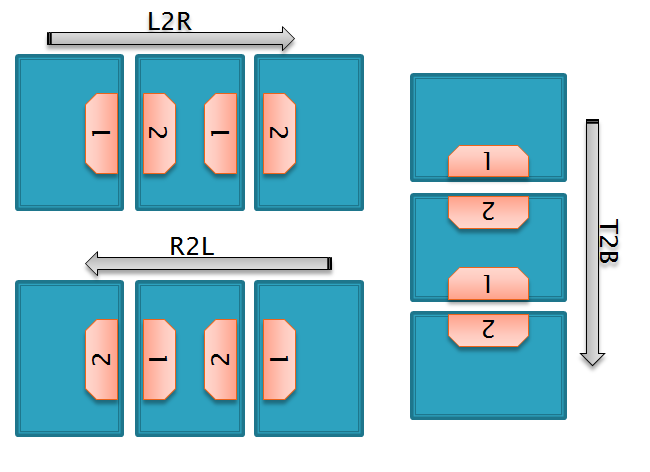
|  |
| --- |
| Sample\_016\_Sealstamp  markProp = new SealstampProperties("classpath:RCSS00001.png");  markProp.setPositionType(PositionType.DEFAULT);  super.setMarkInfos(new PropertiesMarkInfo(markProp)); |

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，通常用於證明文件、申請書表或裝訂輸出。輸出方向以setTextDirection()指定，當輸出在頁面上下時，會把圖片水平對切；輸出於左右時，圖片垂直對切。圖片的前半部稱之為「首章」，會輸出在頁面的尾端；後半部「尾章」則輸出在頁面首端。注意當輸出方向為L2R(左至右)或R2L(右到左)時，首尾章的部分正好相反。

騎縫章的輸出模式由 setPositionType() 指定，可用模式如後述。除此之外，圖片的濃淡、大小、切割比例及擺放中心位置，也都可以經由SealstampProperties的參數進行調整。

#### DEFAULT

未特別設定時的預設值。第一頁不會輸出尾章，最後一頁不會輸出首章，因為沒有對應的另一半可以併起來。但文件只有一頁時是例外，仍舊輸出首章。若只有一頁的時候不要輸出任何騎縫章，可以設定setIgnoreOnePage(true)。



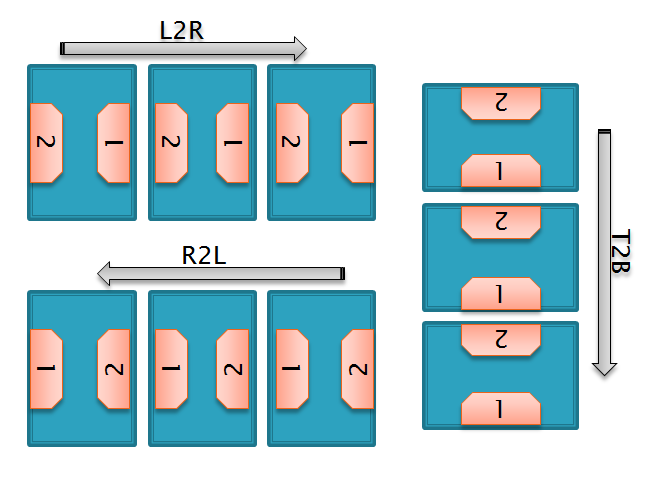
若文件中有分節，則騎縫章輸出的首末頁範圍是以節為單位處理。在每一節開頭，若有不輸出頁碼的封面頁部分，也不會輸出騎縫章。

#### IGNORE\_SECTIONS

輸出原則同DAFAULT，但是忽略文件中的分節資訊，也沒有對於封面頁做特殊處理。

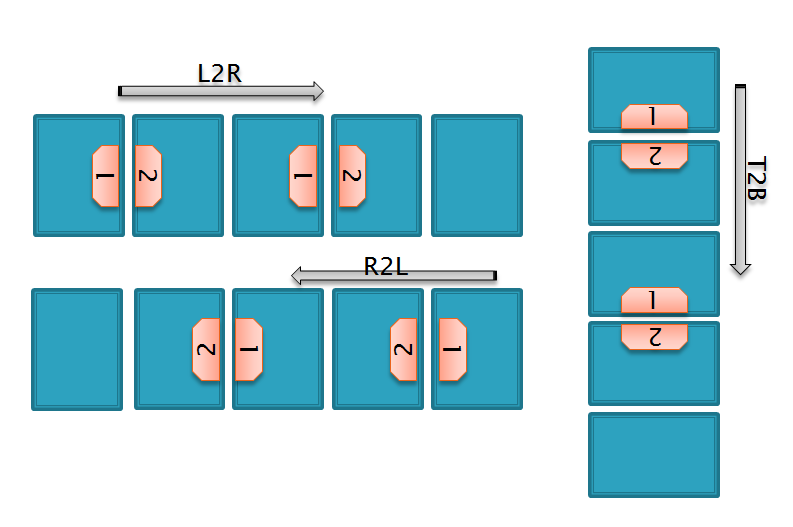
#### FULL

所有頁面均輸出完整首尾章，setIgnoreOnePage設定無效。

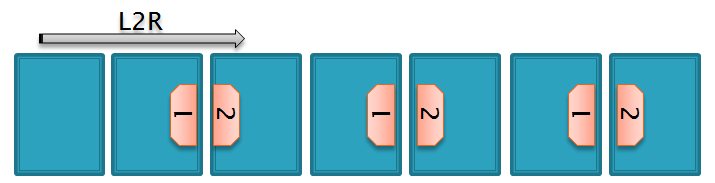


#### BINDING

裝訂模式，每一單頁只會輸出首章或尾章之一。若文件頁數為奇數，因為沒有辦法輸出對應的尾章，則最後一頁的首章會跳過不印。

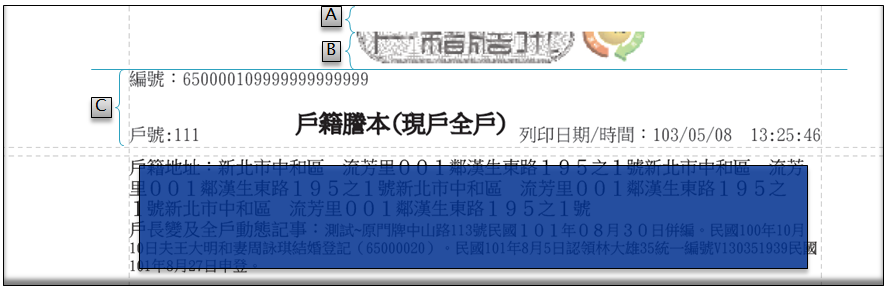


配合一般裝訂，可以增設setSkipPages(1)，以跳過第一頁輸出。



### 計算騎縫章邊界範圍

當文件有輸出騎縫章時，頁面的邊界留白應該要自行計算，以保留騎縫章輸出空間，避免與頁首頁尾重疊。



評估時，分成ABC三部分加總就可以得到應設定留白大小。

* A：印表機輸出邊界，因為各印表機實際會有部分空間無法輸出，所以要把騎縫章切割的邊界對齊輸出在該位置。  
  SealstampProperties定義此預設值為18 pixel。
* B：騎縫章高度，(圖檔高度\*縮放比) / 2 = (0.3 x 140/2) = 21。
* C：希望保持在騎縫章以下的頁首高度。  
  以字型12輸出4行文字，預設頁首行距1，則取 (12+1) x 4=52。

故設定為 18+21+52 = 91。

### 外部設定檔定義

將以上浮水印設定記錄於外部文字設定檔的方案。唯功能實作待調整，暫不隨同套件版本發布。

## 文件保全

以 EncryptionInfo 設定。

待補充文件說明。

# 基本Excel 輸出

## ExcelDocument摘要

待補充文件說明

已知問題：POI 所使用的 XMLBeans 對於非0字面文字輸出有問題。

## 格式設定

## 合併欄位

## 列印設定

待補充文件說明

# 表格與樣式

本套件以CellFormat來定義個別Cell的顯示外觀，在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，就把defaultFormat設定完成。若輸出欄位後，才異動defaultFormat，雖不會影響到已輸出的欄位外觀，但這是不推薦使用的特殊用法，維護性較差。

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

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

|  |
| --- |
| Sample\_101\_CellFormat\_bgcolor  final TableiText table = pdfDocument.createTable(98, 16);  table.getDefaultFormat().setFontSize(8)  for (int r = 0;r<255;r+=16){  for (int g = 0;g<255;g+=16){  for (int b = 0;b<255;b+=16){  final CellFormat cf = new CellFormat()  .setBackgroundColor(new Color(r,g,b));  table.addCell(String.format("%X%X%X",r,g,b), cf);  }  }  }  table.appendMe(); |

## 邊框

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。

|  |
| --- |
| Sample\_101\_CellFormat\_border\_show |

## 文字、對齊

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()、getFontType()、getFontBold()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

|  |
| --- |
| Sample\_101\_CellFormat\_font  final CHTFontFactory risFontFactory = pdfDocument.getFontFactory();  final FontInfo textBgGray10B  = risFontFactory.createFontInfo(10, FontStyle.NORMAL, Color.BLACK);  final FontInfo textBgGray10R  = risFontFactory.createFontInfo(10, FontStyle.UNDERLINE, Color.RED);  table.addCell(  new SubPhrase(textBgGray10B, "多種樣式、特殊字、換行範例\n"),  new SubPhrase(textBgGray10R, "|𠀡|𠀖|" + "第２字面中文\n"),  new SubPhrase(textBgGray10B, "|𠀡|𠀖|\n")  ); |

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：

|  |
| --- |
| Sample\_101\_CellFormat\_align |

文字與邊框間的距離，可以用setPadding系列Method設定。相關的設定值會把先前已設定的值蓋掉。例如先呼叫setPaddingB(b)再呼叫setPadding(p)，則原本設定PaddingB=b就變成p了；若呼叫順序反過來，就是其它三個方向為p，下方為b。

* setPadding = setPaddingH(float) + setPaddingV(float)
* setPaddingH(float) = setPaddingL(float)+setPaddingR(float)
* setPaddingV(float) = setPaddingT(float)+setPaddingB(float)

## 跨欄、跨列與子表格(InnerTable)

在TableiText中，呼叫addCell時直接傳入colspan值，即可達成水平跨欄效果，幾個相關Method如下：

addCell(int, SubPhrase...)

addCell(String, int, SubPhrase...)

addCell(CellFormat, int, SubPhrase...)

|  |
| --- |
| Sample\_100\_TableSpan\_columnSpans  final TableiText table = pdfDocument.createTable(100, 5);  // row1  repeatCells(table, "", 1);  table.addCell("colSpans = 2", 2);  repeatCells(table, "", 2);  // row2  repeatCells(table, "", 2);  table.addCell("colSpans = 3", 3);  table.appendMe(); |

水平跨欄的內部實作，本套件使用PdfPTable，但是垂直跨欄就沒那麼單純。若直接設定rowSpan，IText在處理表格換頁時就會發生問題，以下在 Sample\_100\_TableSpan\_rowspans\_bug示範配合幾種不同PdfPTable表格參數時的異常案例。

|  |
| --- |
| Sample\_100\_TableSpan\_rowspans\_bug  // 第一欄使用 setRowspan 輸出  mrCell.setRowspan(rowspan); |

|  |
| --- |
| 第一欄內容過長 且設定PdfPTable.setSplitRows(true)  第一欄後半部的文字被切斷，沒有完成輸出。 |
|  |
| 第一欄內容過長 且設定PdfPTable.setSplitRows(false  因為不允許分割欄位，所有後半部的欄位都不見了 |
|  |
| 使用本套件包裝的InnerTable進行輸出，左方欄應輸出資料有正常呈現  右方欄完成輸出後，次頁顯示為空白。 |
|  |

使用InnerTable的方式，第一步先由原先的TableiText建立 InnerTable，然後由左至右，跨列的部分用原先的 TableiText輸出；不跨列的部分用 InnerTable輸出並以appendMe()輸出回TableiText。

#### createInnerTable(int leftCols, int rightCols)

適用於大部分的情況，建立時的傳入參數分別定義保留在InnerTable左右，不被InnerTable佔用的欄位數，也就是需要跨列的部分。

輸出完成直接呼叫InnerTable.appendMe()，外部的TableiText 會依據leftCols、rightCols自動計算InnerTable佔用的欄位數所佔用的欄位數。

|  |
| --- |
| Sample\_100\_TableSpan\_rowSpan\_normal  final TableiText table = pdfDocument.createTable(95, 7);  table.addCell("rowSpans = 3");  InnerTable inner1 = table.createInnerTable(1, 3);  repeatCells(inner1, "INNER DATA", 9);  inner1.appendMe();  table.addCell("rowSpans = 3");  InnerTable inner2 = table.createInnerTable(5, 1);  repeatCells(inner2, "INNER DATA", 3);  inner2.appendMe();  table.addCell("rowSpans = 3"); |

#### createInnerTable(float[] subWidth)

若要做出欄位不對齊的效果，就使用這個建構方法。傳入的subWidth表示InnerTable中各子欄寬的比例。最後輸出時，必須使用appendMe(int colspan)，明確指定InnerTable佔用的欄位數。

若要在單一欄位中，切割成多欄輸出，也可以利用此建構方法產出InnerTable繪製。

|  |
| --- |
| Sample\_100\_TableSpan\_rowSpan\_dyna  final TableiText table = pdfDocument.createTable(95, 7);  float[] subWidth = [2, 3, 2, 3, 2, 3, 2]  InnerTable inner2 = table.createInnerTable(subWidth);  repeatCells(inner2, " ", 12);  table.addCell("使用 InnerTable 做出欄位不對齊效果", 2);  inner2.appendMe(3);  table.addCell("", 2); |

如何設定子表格的框線是另一個常見的議題。一般情況，子表格框線會跟著父表格的CellFormat設定樣式輸出。第一個例子是父表格的框線是粗線，但希望子表格內呈現細線：

|  |
| --- |
| Sample\_100\_TableSpan\_rowSpan\_dyna  final TableiText table = pdfDocument.createTable(50, 4);  table.getDefaultFormat().setBorderWidth(1.5f);  // …  table.addCell("");  InnerTable inner1 = table.createInnerTable(1, 1);  inner1.getDefaultFormat().setBorderWidth(0.5f);  for (int i = 0; i < 3; i++) {  inner1.addCell("R",new CellFormat().setAlignH(AlignH.RIGHT));  inner1.addCell("L",new CellFormat().setAlignH(AlignH.LEFT));  }  inner1.appendMe();  table.addCell("");  // …  table.appendMe(); |

在上例中，子表四周的粗線，其實是依據父表格的定義輸出的外框。因為繪製出來的框線會互相疊加，如果我們希望連外框的部分也變成細線，除了子表格本身佔用的欄位外，與其相鄰的欄位也要把外框定義為細線，或是不輸出相鄰格線。同理，若要設定無框線，也要把相鄰外框設為不輸出。

|  |
| --- |
| Sample\_100\_TableSpan\_innerBorder\_innerThinAll  // 把左邊欄位的右框線定義為不輸出  table.addCell("X", new CellFormat().setBorder(Border.NR));  InnerTable inner1 = table.createInnerTable(1, 1);  inner1.getDefaultFormat().setBorderWidth(0.5f);  inner1.setBorderWidth(0.5f);  for (int i = 0; i < 3; i++) {  inner1.addCell("R",new CellFormat().setAlignH(ALIGN.RIGHT));  inner1.addCell("L",new CellFormat().setAlignH(ALIGN.LEFT));  }  inner1.appendMe();  // 把右邊欄位的左框線定義為不輸出  table.addCell("X", new CellFormat().setBorder(Border.NL)); |

|  |
| --- |
| Sample\_100\_TableSpan\_innerBorder\_innerNone  // 承上例，但子表改為設定外框  inner1.getDefaultFormat().setBorder(Border.NONE);  inner1.setBorder(Border.NONE); |

上面範例的設定其實相當的不自然，對於相鄰欄位的格式定義更是容易造成維護上的問題。這樣的表格在設計時，應該考量把具外粗框的區塊逐一視為子表格處理，如下例所示多加一層子表格，這樣的程式風格維護性較佳。

|  |
| --- |
| Sample\_100\_TableSpan\_innerBorder\_innerThinAll\_2  InnerTable tableP = table.createInnerTable(0, 0);  tableP.getDefaultFormat().setBorderWidth(0.5f);  tableP.addCell("X");  InnerTable inner1 = tableP.createInnerTable(1, 1);  for (int i = 0; i < 3; i++) {  inner1.addCell("R",new CellFormat().setAlignH(AlignH.RIGHT));  inner1.addCell("L",new CellFormat().setAlignH(AlignH.LEFT));  }  inner1.appendMe();  tableP.addCell("X");  tableP.appendMe(); |

## 特殊欄位輸出

CellFormat.setCellType()，可提供多樣及客製化的欄位外觀設定。它的實作原理是由對應的CellCreator建立PdfPCell，並使用iText的CellEvent進行外觀的再處理。目前在套件中提供的 CellType 共有兩種，如果要自行開發CellType，應實作一組匹配的CellType/CellCreator類別，並以CelltypeFactory. <T extends Celltype> register(Class<T>, CellCreator<T>)註冊。

### TrisectionTitle

其用途為輸出多分隔標題欄位。呼叫方式與輸出效果請參考範例，當文字過長會超出欄位，請自行估算欄寬或以cellFormat設定字型。

|  |
| --- |
| Sample\_100\_TableSpan\_innerBorder\_innerThinAll\_2  Celltype celltype = new TrisectionTitle(xTitle, contentTitle, yTitle);  table.addCell(new CellFormat().setCellType(cellType));  // addCell 若有傳入其它文字內容，會被忽略。 |

### ExtraLines

用於在欄位中繪製額外的線段，線段資訊以LinePattern進行定義。建構LinePattern時可設定Mode及Border，其中Mode的預設值為Mode.BORDER\_BOX；Border則為必填，宣告此線段適用於欄位四周哪幾個邊上。

Mode的定義有四種，宣告線段兩端的對齊方式，可配合offset參數設定輸出線段的相對位置。

|  |
| --- |
| Sample\_102\_CellType\_Borders  BORDER：線段兩端對齊框線  PADDING：線段兩端對齊欄位留白空間  BORDER\_BOX：線段兩端隨OFFSET縮放，設為0時同BORDER。  PADDING\_BOX：線段兩端隨OFFSET縮放，設為0時同PADDING。 |

#### LinePattern的其它屬性

##### 設定寬度

setWidth(float)

##### 設定顏色

setColor(Color)

##### 設定虛線格式

setDashUnits(float[])指定虛線長度單位，其值為實線、空白、...交錯。若陣列項目數量為奇數，則第二次為反相輸出。

虛線中的第一條實線輸出，可使用setDashPhash(float) 對它做出長度調整。

##### 額外調整長度

只針對Mode.BORDER、Mode.PADDING兩種模式生效：setLengthDx(float)。

#### 範例效果：標題加入特定裝飾線

下例使用兩條線段輸出表格標題的雙底線。

|  |
| --- |
| Sample\_102\_CellType\_DoubleUnderlines  LinePattern line1 =  new LinePattern(LinePattern.Mode.PADDING, Border.B)  .setOffset(0).setLengthDx(1)  .setColor(Color.BLUE);  LinePattern line2 = line1.clone()  .setOffset(-1);  final TableiText table = pdfDocument.createTable(90, 4);  table.getDefaultFormat().setBorder(Border.N)  .setPadding(6).setAlignH(AlignH.JUSTIFIED\_ALL);  CellFormat headerCF = new CellFormat()  .setCellType(new ExtraLines(line1,line2));  table.addCell("HEADER" , headerCF);  table.addCell("HEAD" , headerCF);  table.addCell("HEADER" , headerCF);  table.addCell("HEAD" , headerCF); |

#### 範例效果：虛線表格

虛線外框只能在輸出Cell後繪製，所以原有的欄位需把Border設為無框線才行。

|  |
| --- |
| Sample\_102\_CellType\_DoubleUnderlines  float[] dashUnits = [2f, 2f];  LinePattern baseLinePtn  = new LinePattern(LinePattern.Mode.BORDER, Border.BOX)  .setOffset(0)  .setColor(Color.BLUE)  .setDashUnits(dashUnits);  final TableiText table = pdfDocument.createTable(50, 4);  table.getDefaultFormat().setBorder(0)  .setCellType(new ExtraLines(baseLinePtn));  table.addCell("TEST");  table.addCell("TEST");  table.addCell("TEST",2);  InnerTable inner = table.createInnerTable(0, 1);  inner.addCell("TEST",3);  inner.addCell("TEST",3);  inner.appendMe();  table.addCell("TEST");  table.appendMe(); |

## 表格裝飾

當每一頁的表格繪製完成後，可用自訂的TableiTextDecorator加上表格裝飾效果。套件中也有提供一些常用的的裝飾類別。

### TableHeader

在表格外框之上/下，輸出Header/Footer。TableHeader. setHeader 的設定方式，可參考頁首頁尾的設定說明。但原則上，必須再加上設定table的 spacingAfterInCM/BeforeInCM以保留表首/表尾的輸出空間。

|  |
| --- |
| Sample\_104\_TableDecorator\_basicHeader  tableHeader.setHeader(HeaderPosition.CenterHeader, "表格一標題", 14);  tableHeader.setHeader(HeaderPosition.LeftHeader, "左上1", 10);  tableHeader.setHeader(HeaderPosition.LeftFooter, "左下1", 10);  tableHeader.setHeader(HeaderPosition.RightHeader, "右上1", 10);  tableHeader.setHeader(HeaderPosition.RightFooter, "右下1", 10);  final TableiText table1 = pdfDocument.createTable(80, 3);  repeatCells(table1, "TEST1", this.lines \* 3); //  table1.addEventDecorators(tableHeader);  table1.setSpacingAfterInCM(1);  table1.setSpacingBeforeInCM(1);  table1.appendMe();  // TABLE2  TableHeader tableHeader2 = new TableHeader();  tableHeader2.setHeader(HeaderPosition.CenterHeader, "表格二標題", 14);  tableHeader2.setHeader(HeaderPosition.RightHeader, "右上2", 12);  tableHeader2.setHeader(HeaderPosition.RightFooter, "右下2", 12);  final TableiText table2 = pdfDocument.createTable(80, 3);  repeatCells(table2, "TEST2", this.lines \* 3);  table2.addEventDecorators(tableHeader2);  table2.appendMe(); |

### TableBorder

用於繪製表格外框。外框線段的定義方式，可參考前一節ExtraLines中的相關說明。PADDING／PADDING\_BOX兩個模式在此定義的參考位置，為Table上下由spacingAfterInCM spacingBeforeInCM所定義的表首/表尾輸出空間。

BORDER／BORDER\_BOX即是表格實際輸出空間，如希望跳過headerRows或footerRows，可設定 skipTop或skipBottom，則框線位置會自動做出調整。

|  |
| --- |
| Sample\_104\_TableDecorator\_outerBorder  pdfDocument.writeText("輸出實線外框");  tableWithOuterBorder(pdfDocument, 0);  pdfDocument.writeText("外擴框線");  tableWithOuterBorder(pdfDocument, -3);  pdfDocument.writeText("外擴雙框線");  tableWithOuterBorder(pdfDocument, -3, -5);  pdfDocument.writeText("內縮框線");  tableWithOuterBorder(pdfDocument, 1);  private tableWithOuterBorder(PDFDocument pdfDocument, float offset) {  final TableiText table = pdfDocument.createTable(80, 3);  table.addEventDecorators(new TableBorder(  new LinePattern(borderMode, Border.BOX).setOffset(offset)  ));  // ….  } |

### RowBackgroundColor

設定可每列底色變化，以下示範輸出標題列用一種顏色、內容列用三種顏色輪換。

|  |
| --- |
| Sample\_104\_TableDecorator\_colors  RowBackgroundColor rbc = new RowBackgroundColor(new Color(0xE0F0F0)  , new Color(0xF0F0E0)  , new Color(0xF0E0F0))  final TableiText table = pdfDocument.createTable(80, 3);  table.getDefaultFormat()  .setBorder(Border.NONE)  .setBackgroundColor(Color.lightGray);  repeatCells(table, "HEADER1", 3);  repeatCells(table, "HEADER2", 3); // Header 佔兩列  repeatCells(table, "FOOTER", 3); // Footer 佔一列  table.setHeaderRows(3); // 所有Header 共三列  table.setFooterRows(1); // 其中Footer 佔一列  repeatCells(table, "TEST", 15);  table.addCell("INNER TABLE \n整塊內部表格使用同一底色"  , new CellFormat().setAlignV(AlignV.MIDDLE), 2);  InnerTable inner = table.createInnerTable(2, 0);  repeatCells(inner, "TEST2", 5);  inner.appendMe()  repeatCells(table, "TEST", this.lines\*3);  table.addEventDecorators(rbc);  table.appendMe()  P1.  P2. |

# 通用表格

## 概念

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。

## 設定方式

### 直列表格

直列表格即是一般最常見的報表型式，可有階層關係的標題列在最上方，但每列資料只佔一列。同一列之間的欄位可能有相依關係，如相加、相減。所有資料也可能依前方的幾個欄位做群組統計。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 |

利用本套件輸出此類表格的設定方式，第一步是先建立NormalTableMetadata，其後再插入個別欄位定義(含資料來源及格式)、群組定義。最後再使用PDFNormalTableTransfer、ExcelNormalTableTransfer等表格轉換器，撘配原始資料(List<JavaBean/Map>)轉換為表格輸出。

以下逐一介紹 NormalTableMetadata 的可設定特性。

#### 表格標題(caption)

表格標題，即為當表格換頁時，重複出現在表格上方，無框級的表格名稱。但與前一章所介紹的 TableHeader 略有不同，此處所定義的Caption是利用跨所有欄位的無框線欄位進行輸出。

建立TableMetadata時就可傳入Caption文字。若為空字串就不顯示標題，也不佔用空間。

|  |
| --- |
| Sample\_104\_TableDecorator\_outerBorder  pdfDocument.writeText("無標題表格");  new PDFNormalTableTransfer(pdfDocument, BASIC\_METADATA(""))  .transTable(SAMPLE\_VO.mockDataset());  pdfDocument.newPage();  pdfDocument.writeText("含標題表格：(標題：測試表格)");  new PDFNormalTableTransfer(pdfDocument, BASIC\_METADATA("標題：測試表格"))  .transTable(SAMPLE\_VO.mockDataset());  public NormalTableMetadata BASIC\_METADATA(String captionText) {  final NormalTableMetadata tm = new NormalTableMetadata(captionText);  tableMeta.getDefaultContentFormat().setAlignV(AlignV.MIDDLE);  ColumnMetadata colA = tm.append("年度", new BeanProperty("text1"), 20);  ColumnMetadata colB = tm.append("地區", new BeanProperty("text2"), 20);  ColumnMetadata colC = tm.append("項目", new BeanProperty("text3"), 20);  ColumnMetadata colD = tm.append("值1", new BeanProperty("value1"), 20);  colD.getContentFormat().setAlignH(AlignH.RIGHT);  return tm;  }  // 輸出結果： |

#### 加入欄位

每一個欄位的定義，是用一個ColumnMetadata 物件表示，其中資訊包含標頭文字、欄寬、資料來源及顯示格式。對於NormalTableMetadata 或 ColumnMetadata，可以使用append 或 prepend method 在當下位置加入子欄位並取回新的 ColumnMetadata 物件。append/prepend 方法的參數為 String title(標頭), CellDataSource field(資料來源), float width(欄寬) 三個項目的排列組合。顯示格式則是利用它的 getContentFormat() / getHeaderFormat() / setStyle(String) 等method 加以定義。

title(標頭)為顯示在Header列的文字內容，當表格發生換頁時，會重複顯示。

field(資料來源)定義如何取得內容值，如由javaBean的指定欄位取值、常數值、相依計算等等…。

width(欄寬)值表示欄寬比重。但一個ColumnMetadata之下仍有子欄位時，它本身的本身的欄寬比重設定無效，會改採下一層欄位的總合計算。

#### 顯示格式定義

所有的顯示格式，同樣是經由CellFormat這個類別進行定義。TableMetadata 可取得DefaultHeaderFormat 及DefaultContentFormat，這兩項分別可定義預設的標題列及內容列樣式。ColumnMetadata可設定 StyleName名稱，由TableMetadata.getStyle(styleName)可取得對應的ColunmStyle物件。ColunmStyle.getHeader() / getContent() 即是相對的標題列及內容列樣式。若要獨立設定單一ColumnMetadata中的樣式，則使用getContentFormat() / getHeaderFormat() 進行修改。

最後進行表格轉換時，則由ColumnMetadata樣式、Sytle樣式、Default樣式優先順序取得非NULL 的定義值。

在 CellFormat 的屬性textFormat，在一般產製表格時沒有作用，但在通用表格轉換時，若傳入的物件為數值或日期，則會試著套用相關格式轉為字串輸出。數值格式依DecimalFormat為準；日期格式依SimpleDateFormat為準。

#### 資料來源定義

##### BeanProperty

執行transTable(…)時，必須傳入一個Object[] 或 List<Object>，在其中的每一個元素，應帶有單一列的所有資料內容。設定 BeanProperty，即是以 apache.commons 的PropertyUtils. getProperty 對其取值。取回的值會是Object，顯示在內容列的文字即是其toSting() 的回傳值。

##### 格式轉換

取得的 BeanProperty 若原始型態為字串，但想要轉換為數值或日期格式輸出；或反之數值資料直接用字串輸出。可以使用 .asInteger() .asText() 之類的方法，取得Transfer實作。

下例將預設的 textFormat 設為 “#,##0”，所以原先為數值資料的value3，或使用asInteger轉換過的text4，都會轉為此格式輸出。

|  |
| --- |
| Sample\_200\_TableMetadata\_BASIC\_FromBean  final NormalTableMetadata tm  = new NormalTableMetadata("欄位格式、型態支援展示");  tm.getDefaultContentFormat()  .setAlignV(AlignV.MIDDLE)  .setTextFormat("#,##0");  ColumnMetadata colA1  = tm.append("Text4 As Object ", new BeanProperty("text4"), 60);  ColumnMetadata colA2  = tm.append("Text4 As String ", new BeanProperty("text4").asText(), 60);  ColumnMetadata colA3  = tm.append("Text4 As Integer", new BeanProperty("text4").asInteger(), 60);  ColumnMetadata colB1  = tm.append("Value3 As Object ", new BeanProperty("value3"), 60);  ColumnMetadata colB2  = tm.append("Value3 As String ", new BeanProperty("value3").asText(), 60);  ColumnMetadata colB3  = tm.append("Value3 As Integer", new BeanProperty("value3").asInteger(), 60);  new PDFNormalTableTransfer(pdfDocument, tm)  .transTable(SAMPLE\_VO.mockDataset()); |

在日期部分，預設的輸出格式是yyyy/MM/dd。同樣的，也可以使用asDate/asDatetime 將字串轉為Date物件。可以參考Sample\_200\_TableMetadata\_BASIC\_FromBean\_ToDate範例。另外，對於民國年月日的支援，可以使用

ROCDate(CellDataSource)

ROCDate(CellDataSource, String)

只要讀入的欄位資料為Date型別，或者是YYYMMDD的七碼民國年月日字串，就會使用twy/MM/dd或其它自訂格式輸出。

|  |
| --- |
| Sample\_200\_TableMetadata\_BASIC\_FromBean\_ROCDate  final ColumnMetadata g3 = tableMeta.append("轉為民國格式 (twy/MM/dd)");  g3.append("date1", new ROCDate(new BeanProperty("date1")), 40);  g3.append("date2", new ROCDate(new BeanProperty("date2")), 40);  g3.append("twy/M/d", new ROCDate(new BeanProperty("date2"), "twy/M/d"), 60);  g3.append("yyyy/M/d", new ROCDate(new BeanProperty("date2"), "yyyy/M/d"), 60);  g3.append("民國", new ROCDate(new BeanProperty("date2"), "民國y年M月d日"), 60);  new PDFNormalTableTransfer(pdfDocument, tableMeta)  .transTable(SAMPLE\_VO.mockDataset(100)); |

##### 常數值

有時我們會需要輸出固定的文字或流水號在某個欄位。這時可以用new Const(Object obj) 輸出常值；用new Counter( int start) 輸出自動編號。

|  |
| --- |
| Sample\_200\_TableMetadata\_BASIC\_Const\_Value  tableMeta.append("CONST:Text", new Const("text1"), 30);  tableMeta.append("CONST:Text", new Const("1024"), 30);  tableMeta.append("CONST:Number", new Const(1024), 30);  tableMeta.append("Counter starts with 0", new Counter(0), 30);  tableMeta.append("Counter starts with 1", new Counter(1), 30); |

##### 公式

當column 之間有相依關係，但JavaBean中未實作相關計算時，可以使用公式相關設定。使用公式時，參照對象皆為ColumnMetadata。首先使用ColNumber取得第一個內容值，其型別為NumberOP，接著可以使用它的加減乘除Method，設定與其它ColnumMetadata 間的相依公式。

|  |
| --- |
| Sample\_200\_TableMetadata\_BASIC\_Const\_Value  tableMeta.append("CONST:Text", new Const("text1"), 30);  tableMeta.append("CONST:Text", new Const("1024"), 30);  tableMeta.append("CONST:Number", new Const(1024), 30);  tableMeta.append("Counter starts with 0", new Counter(0), 30);  tableMeta.append("Counter starts with 1", new Counter(1), 30); |

##### 其它

集合資料合併為字串使用ConcatString。

多來源資料格式化輸出使用MultipleSources。

條碼資料使用BarcodeImage。

當原始資料的每一個列元素是Object[]時，使用ArrayData。

### 清冊表單

當一筆資料可能需要多列才足以輸出完整內容時，在此定義為清冊表單，需要使用NestTableMetadata進行巢狀描述定義。以下用一個典型的表冊範例示範如何描述巢狀表格。

#### 基本清冊

|  |
| --- |
| Sample\_202\_Nest\_BASIC\_1  完整結果如下：    範例程式逐步說明  // 開始設定 TableMETADATA  final NestTableMetadata tableMeta = new NestTableMetadata("LIST");  tableMeta.setDefaultColumnWidth(10); // 預設欄寬比重為10  tableMeta.getDefaultHeaderFormat()  .setFontBold(true) // 標題粗體  .setFontSize(14) // 字型14  .setMinHeightInCM(1); // 列高設為 1CM  tableMeta.getDefaultContentFormat()  .setAlignV(AlignV.MIDDLE); // 內容文字垂直置中  tableMeta.setBorderWidth(1.5f); // 每一組資料的外框設為粗線  // 開始加入第一列的METADATA  tableMeta.append("年度", new BeanProperty("text1"));  tableMeta.append("地區", new BeanProperty("text2"), 20);  tableMeta.append("項目", new BeanProperty("text3"), 20);  // 定義第一列的欄位項目，其寛度比例為 10:20:20    // 第二列開始，先換列  tableMeta.nextRow();  // ColumnMetadata 的 append/prepend行為與直列表格略有不同  // 加入的子欄位同樣繼承本欄位的樣式屬性，  // 但是顯示位置為接續在該欄之後  final ColumnMetadata col1 = tableMeta.append("值1", new BeanProperty("value1"));  col1.getContentFormat().setAlignH(AlignH.RIGHT).setTextFormat("#,##0");  col1.append("值2", new BeanProperty("value2"));  col1.append("值3", new BeanProperty("value3"), 40);    // 在第二列中，加入垂直分割的兩個欄位：開始日期、結束日期  // 所以前面三個欄位的高度自動被拉大。  final NestTableMetadata sub1 = tableMeta.newSubTableVertical(  new SubColumnParameter("開始日期", new BeanProperty("date2"), 40),  new SubColumnParameter("結束日期", new BeanProperty("date1"), 30)  );  // 此例中故意把寬度設為不同，此時取最大值 40 為其欄寬比重  // 所以第二列的欄位項目，其寛度比例為 10:10:40:40    // 完成轉換動作  PDFNestTableTransfer transfer = new PDFNestTableTransfer(pdfDocument, tableMeta);  transfer.transTable(SAMPLE\_VO.mockDataset()); |

#### 無框線欄位顯示

以下示範如何在表格前方加上無框線的勾稽欄位。

|  |
| --- |
| Value  // 開始設定 TableMETADATA  final NestTableMetadata outerTable = new NestTableMetadata("");  outerTable.setWidthPercentage(95);  outerTable.setDefaultColumnWidth(0.3f);  outerTable.getDefaultFormat().setBorder(Border.N); // 最外層表格設定為無框線  outerTable.getDefaultContentFormat().setAlignV(AlignV.MIDDLE);  // 加入無框線的勾稽欄位  outerTable.append("", Const.define("V"));    // 具粗框的主要內容，建立一個子 NestTable處理。  addAllMainColnum(outerTable.newSubTable());  處理主表格的Method：addAllMainColnum()  private void addAllMainColnum(final NestTableMetadata mainTable) {  // 外框加上粗線  mainTable.setBorder(Border.A);  mainTable.setBorderWidth(3f);    // 內框設為輸出所有框線並HEADER置左  mainTable.getDefaultFormat().setBorder(Border.A);  mainTable.getDefaultHeaderFormat().setAlignH(AlignH.LEFT);    // 為序號欄位定義 Style : 定義為全部置中  ColunmStyle styleCenter = mainTable.getStyle("CENTER");  styleCenter.getHeader().setAlignH(AlignH.CENTER).setAlignV(AlignV.MIDDLE);  styleCenter.getContent().setAlignH(AlignH.CENTER).setAlignV(AlignV.MIDDLE);  mainTable.append("序號", new Counter(1)).setStyle("CENTER");    // 次欄一次加入4個垂直欄位  mainTable.subTableWith( //  new SubColumnParameter("姓名", Const.define(""), 4), //  new SubColumnParameter("出生日期", Const.define(""), 4), //  new SubColumnParameter("統號", Const.define(""), 4), //  new SubColumnParameter("連絡電話", Const.define(""), 4));    // 再用一個子表格處理接下來的欄位  final NestTableMetadata table2 = mainTable.subTable();    // 第一列：至遷出地  table2.append("遷入日期\n時：分：秒", Const.define(""), 4);  table2.append("遷入別", Const.define(""), 4);  table2.subTableWith( //  new SubColumnParameter("戶長統號", Const.define(""), 4), //  new SubColumnParameter("戶長姓名", Const.define(""), 4));  table2.append("遷出地", Const.define(""));    // 第二列：  table2.nextRow();  table2.subTableWith( //  new SubColumnParameter("出生地", Const.define(""), 10), //  new SubColumnParameter("學歷", Const.define(""), 10));  table2.append("遷入地", Const.define(""), 11.5f);    } |

#### 巢狀清冊的樣式繼承

表格樣式依以下順序繼承，下方的非NULL設定會覆蓋上層的設定值。其中Style設定是所有NestTableMetadata階層共用。

* Parent.defaultFormat
* Child.defaultFormat
* Style.[Header/Content]Format
* Parent.default[Header/Content]Format
* Child. default[Header/Content]Format
* Cell.headerFormat/ContentFormat

#### 輸出為CSV

當使用 CSVNestTableTransfer 處理巢狀清冊時，會依欄位加入順序，將所有資料輸出為直列表格，以便第三方程式進行加值處理。

## 特殊輸出

有時業務需求會在基本的通用表格上做出變化，例如當報表欄位不多時，會讓同一頁面用分欄的方式，盡可能呈顯示多組資料內容。又或者當欄位太多時，塞不進同一頁面，會把欄位以水平切割方式，逐頁輸出。

以下是針對這些特別需求的通用實作，但有特別設定表格框線時，可能輸出會不如預期。

### 多欄表格

#### 直列優先

先輸出直列至頁尾，再換下一直欄輸出。預設為每頁重複 2 直欄；每一直欄的外框加粗；在結尾輸出時，會以灰線補齊所有框線。利用 fillToEnd 屬性可設定是否補齊框線，extendBlocks設定是否把直欄外框擴展至整頁範圍。

|  |
| --- |
| Sample\_203\_DuplicatedTable\_BASIC\_VFirst  // 使用 BASIC\_METADATA("") 建立每一直欄所用的 NormalTableMetadata 。  new PDFDuplicatedTableTransfer(pdfDocument, BASIC\_METADATA(""), Mode.V\_FIRST)  .setRepeat(2) // 每頁重複 2 直欄  .transTable(SAMPLE\_VO.mockDataset(115));  P1: P2: |

|  |
| --- |
| Sample\_203\_DuplicatedTable\_BASIC\_VFirst\_NotFill  // 使用 BASIC\_METADATA 建立每一直欄所用的 NormalTableMetadata 。  new PDFDuplicatedTableTransfer(pdfDocument,  BASIC\_METADATA("標題：　測試表格"), Mode.V\_FIRST)  .setRepeat(3) // 每頁重複 3 直欄  .setFillToEnd(false) //  .transTable(SAMPLE\_VO.mockDataset(115));  P1: P2: |

#### 橫欄優先

所有資料依水平順序於各直欄輪流輸出。預設為每頁重複2 直欄；每一直欄的外框加粗；在結尾輸出時，會以灰線補齊所有框線，並將外框拉至頁尾。同樣可用 fillToEnd 屬性設定是否補齊框線，extendBlocks設定是否把直欄外框擴展至整頁範圍。

|  |
| --- |
| Sample\_203\_DuplicatedTable\_BASIC\_HFirst  new PDFDuplicatedTableTransfer(pdfDocument,  BASIC\_METADATA("標題：　測試表格"), Mode.H\_FIRST)  .setRepeat(5) // 每頁重複 5 直欄  .transTable(SAMPLE\_VO.mockDataset(105));    // 使用 BASIC\_METADATA 建立每一直欄所用的 NormalTableMetadata 。  new PDFDuplicatedTableTransfer(pdfDocument,  BASIC\_METADATA("標題：　測試表格"), Mode.H\_FIRST)  .setRepeat(5) // 每頁重複 5 直欄  .setExtendLastBlock(false) // 不擴展直欄至整頁  .transTable(SAMPLE\_VO.mockDataset(105)); |

### 欄位跨頁表格

當欄位太多時，塞不進同一頁面，會把欄位以水平切割方式，逐頁輸出的案例。原理是利用TableEventHandler於輸出TABLE 前後進行處理。

元件架構待重新整理，暫不隨同套件版本發布。

### 大量表格

針對表格內容量過大，需一邊取得內容，一邊輸出至OutputStream，避免OutOfMemory的案例。

元件架構待重新整理，暫不隨同套件版本發布。

# 其它議題

## PaintTool

用於精準輸出圖形、文字於定位的工具類別。需配合Coordinate建構實例，預設已有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)

## 條碼輸出

以Coordinate配合建立Barcode39Drawer、Barcode128Drawer、QRCodeDrawer。再使用draw(PdfContentByte contentByte, String text, PointF pos, float w, float h, boolean showText)即可於指定位置輸出指定大小的條碼。

## PageEvent

使用IText原生的PdfPageEventHelper處理，使用時機多半用在實作onEndPage以自行處理頁首/尾輸出。如果有頁次資訊控制邏輯較複雜的情況，可能會需要改用PageEvent自行處理。

