Apache POI: Cómo definir estilos en archivos excel

Publicado el **21 August, 2015** por **Ángel** —

Cómo crear un estilo de celda

La clase que representa un estilo de una celda es `HSSFCellStyle`. Nos permite definir características propias de la celda: tipo de borde, color del borde, alineación del contenido, color de fondo, etc. Para crear un estilo de celda, solo necesitamos una referencia al libro de trabajo de excel:

```
HSSFWorkbook workbook = new HSSFWorkbook();
HSSFCellStyle style = workbook.createCellStyle();
```

Una vez creado el estilo, podemos invocar sus métodos `set...()`, para configurar cada una de sus propiedades. La clase `HSSFCellStyle` posee una serie de propiedes definidas que nos ayudarán a configurar nuestro estilo. Así mismo, la clase que nos permite definir los colores utilizados en nuestros estilos es `HSSFColor`:

```
// Sets the cell background
style.setFillForegroundColor(HSSFColor.BLUE_GREY.index);
style.setFillPattern(HSSFCellStyle.SOLID_FOREGROUND);
```

Probablemente, también querremos modificar las propiedades del texto que se muestra en las celdas. Para hacer esto, debemos crear un estilo de fuente (a continuación).

Cómo crear un estilo de fuente

Los estilos de fuente está representados por la clase `HSSFFont`. Esta clase nos da control sobre la fuente tipográfica usada, color del texto, negrita, cursiva, etc. Al igual que los estilos de celdas, podemos crear estilos de fuentes a partir del libro de trabajo:

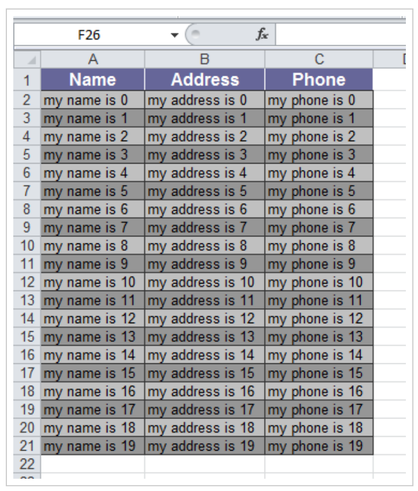
```
HSSFFont font = workbook.createFont();
```

Una vez tenemos la fuente, la asignamos al estilo donde queremos aplicarla:

```
style.setFont(font);
```

Ejemplo Completo

Como ejemplo, vamos a generar un archivo excel tal que así:



	A	B	C
1	Name	Address	Phone
2	my name is 0	my address is 0	my phone is 0
3	my name is 1	my address is 1	my phone is 1
4	my name is 2	my address is 2	my phone is 2
5	my name is 3	my address is 3	my phone is 3
6	my name is 4	my address is 4	my phone is 4
7	my name is 5	my address is 5	my phone is 5
8	my name is 6	my address is 6	my phone is 6
9	my name is 7	my address is 7	my phone is 7
10	my name is 8	my address is 8	my phone is 8
11	my name is 9	my address is 9	my phone is 9
12	my name is 10	my address is 10	my phone is 10
13	my name is 11	my address is 11	my phone is 11
14	my name is 12	my address is 12	my phone is 12
15	my name is 13	my address is 13	my phone is 13
16	my name is 14	my address is 14	my phone is 14
17	my name is 15	my address is 15	my phone is 15
18	my name is 16	my address is 16	my phone is 16
19	my name is 17	my address is 17	my phone is 17
20	my name is 18	my address is 18	my phone is 18
21	my name is 19	my address is 19	my phone is 19
22			

Utilizaremos una clase auxiliar llamada `FakeDataProvider`, que simplemente nos devuelve de datos de prueba para rellenar la tabla:

```
public final class FakeDataProvider {

    /** Return the columns name for the table */
    public static List<String> getTableHeaders() {
        List<String> tableHeader = new ArrayList<String>();
        tableHeader.add("Name");
        tableHeader.add("Address");
        tableHeader.add("Phone");

        return tableHeader;
    }


    /**
     * Return values for the table
     *
     * @param numberOfRows Number of rows we want to receive
     *
     * @return Values
     */
    public static List<List<String>> getTableContent(int numberOfRows) {
        if (numberOfRows <= 0) {
            throw new IllegalArgumentException("The number of rows must be a positive integer");
        }

        List<List<String>> tableContent = new ArrayList<List<String>>();

        List<String> row = null;
        for (int i = 0; i < numberOfRows; i++) {
            tableContent.add(row = new ArrayList<String>());
            row.add("my name is " + i);
            row.add("my address is " + i);
            row.add("my phone is " + i);
        }

        return tableContent;
    }

}
```

La clase encargada de la generación del excel es `ExcelGenerator`. Tiene un sólo método público, `generateExcel()`, que devuelve el excel generado:

```
public class ExcelGenerator {

        // Excel work book
        private HSSFWorkbook workbook;

        // Fonts
        private HSSFFont headerFont;
        private HSSFFont contentFont;

        // Styles
        private HSSFCellStyle headerStyle;
        private HSSFCellStyle oddRowStyle;
        private HSSFCellStyle evenRowStyle;

        // Integer to store the index of the next row
        private int rowIndex;


        /**
         * Make a new excel workbook with sheet that contains a stylized table
         *
         * @return
         */
        public HSSFWorkbook generateExcel() {

                // Initialize rowIndex
                rowIndex = 0;

                // New Workbook
                workbook = new HSSFWorkbook();

                // Generate fonts
                headerFont  = createFont(HSSFColor.WHITE.index, (short)12, true);
                contentFont = createFont(HSSFColor.BLACK.index, (short)10, false);

                // Generate styles
                headerStyle  = createStyle(headerFont,  HSSFCellStyle.ALIGN_CENTER, HSSFColor.BLUE_G
                oddRowStyle  = createStyle(contentFont, HSSFCellStyle.ALIGN_LEFT,   HSSFColor.GREY_2
                evenRowStyle = createStyle(contentFont, HSSFCellStyle.ALIGN_LEFT,   HSSFColor.GREY_4

                // New sheet
                HSSFSheet sheet = workbook.createSheet("Very Cool Sheet");

                // Table header
                HSSFRow      headerRow    = sheet.createRow( rowIndex++ );
                List<String> headerValues = FakeDataProvider.getTableHeaders();

                HSSFCell headerCell = null;
                for (int i = 0; i < headerValues.size(); i++) {
                        headerCell = headerRow.createCell(i);
                        headerCell.setCellStyle(headerStyle);
                        headerCell.setCellValue( headerValues.get(i) );
                }


                // Table content
                HSSFRow  contentRow  = null;
                HSSFCell contentCell = null;

                // Obtain table content values
                List<List<String>> contentRowValues = FakeDataProvider.getTableContent(20);
                for (List<String> rowValues : contentRowValues) {
```

```
                        // At each row creation, rowIndex must grow one unit
                        contentRow = sheet.createRow( rowIndex++ );
                        for (int i = 0; i < rowValues.size(); i++) {
                                contentCell = contentRow.createCell(i);
                                contentCell.setCellValue( rowValues.get(i) );

                                // Style depends on if row is odd or even
```

```
                // Autosize columns
                for (int i = 0; i < headerValues.size(); sheet.autoSizeColumn(i++));

                return workbook;
        }


        /**
         * Create a new font on base workbook
         *
         * @param fontColor       Font color (see {@link HSSFColor})
         * @param fontHeight      Font height in points
         * @param fontBold        Font is boldweight (<code>true</code>) or not (<code>false</code>)
         *
         * @return New cell style
         */
        private HSSFFont createFont(short fontColor, short fontHeight, boolean fontBold) {

                HSSFFont font = workbook.createFont();
                font.setBold(fontBold);
                font.setColor(fontColor);
                font.setFontName("Arial");
                font.setFontHeightInPoints(fontHeight);

                return font;
        }


        /**
         * Create a style on base workbook
         *
         * @param font            Font used by the style
         * @param cellAlign       Cell alignment for contained text (see {@link HSSFCellStyle})
         * @param cellColor       Cell background color (see {@link HSSFColor})
         * @param cellBorder      Cell has border (<code>true</code>) or not (<code>false</code>)
         * @param cellBorderColor Cell border color (see {@link HSSFColor})
         *
         * @return New cell style
         */
        private HSSFCellStyle createStyle(HSSFFont font, short cellAlign, short cellColor, boolean c

                HSSFCellStyle style = workbook.createCellStyle();
                style.setFont(font);
                style.setAlignment(cellAlign);
                style.setFillForegroundColor(cellColor);
                style.setFillPattern(HSSFCellStyle.SOLID_FOREGROUND);

                if (cellBorder) {
                        style.setBorderTop(HSSFCellStyle.BORDER_THIN);
                        style.setBorderLeft(HSSFCellStyle.BORDER_THIN);
                        style.setBorderRight(HSSFCellStyle.BORDER_THIN);
                        style.setBorderBottom(HSSFCellStyle.BORDER_THIN);

                        style.setTopBorderColor(cellBorderColor);
                        style.setLeftBorderColor(cellBorderColor);
                        style.setRightBorderColor(cellBorderColor);
                        style.setBottomBorderColor(cellBorderColor);
                }

                return style;
        }
}
```

Ya sólo nos quedaría utilizarlo desde la clase principal y escribir el excel generado en disco:

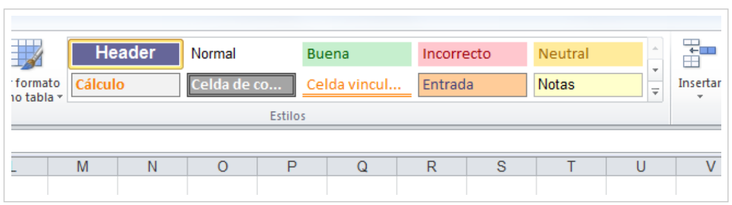
```
public class App {

    public static void main(String[] args) {

        HSSFWorkbook workbook = new ExcelGenerator().generateExcel();

        // Writing the excel to output file
        try {
            OutputStream out = new FileOutputStream("src/main/resources/ExcelWithStyles.xls");
            workbook.write(out);
            workbook.close();
            out.flush();
            out.close();
        }
        catch (IOException e) {
            System.err.println("Error at file writing");
            e.printStackTrace();
        }
    }
}
```

Estilos con nombre

Si definimos un nombre de usuario para el estilo, éste aparecerá entre los estilos predefinidos al abrir el documento desde Microsoft Excel. Por ejemplo, si hubiéramos hecho lo siguiente:

```
headerStyle.setUserStyleName("Header");
```

Nuestro estilo estaría disponible para el usuario:



Estilos sobre filas

Al igual que las celdas, las filas (`HSSFRow`) tienen un método `setRowStyle()`. Es importante tener en cuenta que si aplicamos un estilo sobre una fila, éste sólo se aplica a las celdas vacías. Es decir, que en cuanto creemos una celda sobre esa fila invocando a sú método `createCell()`, ésta no usará el estilo de la fila.

Optimización

Tanto los estilos (`HSSFCellStyle`) como las fuentes (`HSSFFont`) son creados a nivel de libro excel.

Es necesario tenerlo en cuenta porque, si necesitamos aplicar un mismo estilo a varias celdas, lo correcto es crearlo una sola vez y aplicarlo a todas las celdas. Si en lugar de eso, creamos el estilo de nuevo cada vez que tenemos que aplicarlo a una celda, estaremos multiplicando el estilo tantas veces como celdas haya. Esto no es apreciable al mostrar el archivo excel generado, ya que todos los estilos son iguales, pero sí que tendremos un archivo generado de mayor tamaño.

De igual forma, si queremos aplicar el mismo estilo de texto a varios estilos de celda, lo correcto es utilizar una misma fuente para todos los estilos, en lugar de crear una nueva fuente para cada estilo que la utilice.

Código fuente

Puedes descargar el proyecto de ejemplo mi repositorio de GitHub. El ejemplo mostrado corresponde a *Poi-ExcelWithStyles*.

Saludos,
Esta entrada se ha publicado en **Apache POI**, **Excel** y etiquetado como **apache poi**, **excel**, **java** por **Ángel**. Guarda **enlace permanente [https://www.pixnbgames.com/blog/excel/apache-poi-como-definir-estilos-en-archivos-excel/]** .

8 comentarios en «Apache POI: Cómo definir estilos en archivos excel»

Sandra
el **19 febrero, 2016 a las 9:21 pm** ha dicho:

Hola, buenas tardes.

Estoy generando archivos xls con la libreria npoi en visual basic .net, pero cuando genero un archivo con varias hojas y les doy estilo, en algunas ya no quiere ponerlo, sabes a que se debe esta situación?

Gracias

Ángel
el **21 febrero, 2016 a las 10:27 am** ha dicho:

Hola Sandra,

Es complicado saber exactamente la causa del error sin ver el código, pero te aconsejo que revises:

- Que no estás creando un estilo por cada celda, sino que aplicas el mismo a todas las celdas iguales
- Que no estás aplicando un estilo sobre una fila, y luego creando celdas en esa fila. Las celdas creadas no recibirán el estilo de su fila

Siento no poder darte más ayuda en este momento

Sandra
el **22 febrero, 2016 a las 4:11 pm** ha dicho:

Hola, buenos días, gracias por responder.

Ya revisé lo que me sugeriste y al parecer todo esta bien, hice una prueba generando archivos por separado en vez de uno solo con hoja y cuando los crea por separado todo el formato lo pone muy bien, pero al momento de quererlo generar por hojas como que pierde los estilos y las fuentes.
Ahora la idea es generarlos por separado y concatenar esos archivos en uno solo pero no lo he logrado este es el código:

```
Dim CnH As Integer
Dim ArchivoFinal As New HSSFWorkbook
Dim ArchivoFinal2 As New HSSFWorkbook
Dim ArchivoH As HSSFWorkbook
Dim sheetCH As HSSFSheet
Dim file2 As FileStream
Try
For CnH = 1 To totalH
ArchivoH = New HSSFWorkbook(New FileStream(RutaPrincipal & «Temp\\» & filename & «_» & CnH & «.xls», FileMode.Open))
sheetCH = ArchivoH.GetSheetAt(0)
sheetCH.CopyTo(ArchivoFinal, sheetCH.SheetName, True, True)
Next CnH
file2 = New FileStream(RutaPrincipal & «Temp\\» & filename & «.xls», FileMode.Create, FileAccess.ReadWrite)
ArchivoFinal.Write(file2)
file2.Close()
Catch ex As Exception
GrabaError(ex.ToString())
End Try
```

me da el error «Referencia a objeto no establecida como instancia de un objeto»
Esto lo marca en la línea ArchivoFinal.Write(file2)

Sabes a que se debe esta situación?

Ruben
el **11 marzo, 2016 a las 1:13 pm** ha dicho:

Buenos días:
Me gustaría saber si se puede poner dos estilos distintos de letra dentro de la misma celda (una palabra en cursiva y otra en negrita).
En caso de que sí. ¿Como se hace?

Gracias

Ángel
el **25 marzo, 2016 a las 10:24 am** ha dicho:

Hola Rubén,

No puedo ponerte un ejemplo completo en este momento, pero sí que es posible hacer lo que dices. Tendrás que poner como contenido de esa celda un `RichTextString`. En ese objeto es posible definir distintos formatos para partes de un mismo texto.

Gracias por comentar y un saludo,

abel
el **2 junio, 2017 a las 9:14 am** ha dicho:

Buenas, estoy generando un Excel desde java, quiero ajustar las columnas a en relación a su contenido. Uso la función:
hoja.autoSizeColumn((short)0)
el tema es que con el mismo código funciona en el pc de mi compañero, generándose el tamaño correcto, pero con mi pc en mi Excel no hace caso, y se quedan sin ajustar las columnas.
Que puede pasar? Supongo que debe ser la configuración del Excel, pero no doy con la tecla correcta.

Un saludo.

Abel

Gustavo
el **1 agosto, 2018 a las 6:35 pm** ha dicho:

Buenas tardes, no sé si sigue activo este tema, pero tengo una duda sobre como combinar 2 columnas que se encuentren en filas distintas, una debajo de la otra, en realidad lo hago con : hoja.addMergedRegion(new CellRangeAddress(0,1,0,2));
los 2 primeros parametros son las filas, y los otros 2 son las columnas que se quiere combinar, eso lo entiendo, pero en este ejemplo especifico solo combina las celdas de la primera columna, si yo necesito combinar 2 celdas una debajo de la otra de la columna 5.
En este momento sería de mucha ayuda.

Jorge Gomez
el **28 julio, 2020 a las 8:12 pm** ha dicho:

La columna 5 es inidice 4 harías

hoja.addMergedRegion(new CellRangeAddress(0,1,5,5));