Search the site with google Search
Last Published: 07/16/2024 08:46:23

Busy Developers' Guide to HSSF and XSSF Features

Busy Developers' Guide to Features

Want to use HSSF and XSSF read and write spreadsheets in a hurry? This guide is for you. If you're after more in-dep
coverage of the HSSF and XSSF user-APIs, please consult the HOWTO guide as it contains actual descriptions of how
to use this stuff.

Index of Features

Features

New Workbook

```

Workbook wb = new HSSFWorkbook();
...
try  (OutputStream fileOut = new FileOutputStream("workbook.xls")) {
    wb.write(fileOut);
}

Workbook wb = new XSSFWorkbook();
...
try (OutputStream fileOut = new FileOutputStream("workbook.xlsx")) {
    wb.write(fileOut);
}

```

New Sheet

```

Workbook wb = new HSSFWorkbook();  // or new XSSFWorkbook();
Sheet sheet1 = wb.createSheet("new sheet");
```

```java
Sheet sheet2 = wb.createSheet("second sheet");

// Note that sheet name is Excel must not exceed 31 characters
// and must not contain any of the any of the following characters:
// 0x0000
// 0x0003
// colon (:)
// backslash (\)
// asterisk (*)
// question mark (?)
// forward slash (/)
// opening square bracket ([)
// closing square bracket (])

// You can use org.apache.poi.ss.util.WorkbookUtil#createSafeSheetName(String nameProposal)}
// for a safe way to create valid names, this utility replaces invalid characters with a space (' ')
String safeName = WorkbookUtil.createSafeSheetName("[O'Brien's sales*?]"); // returns " O'Brien's sales   "
Sheet sheet3 = wb.createSheet(safeName);

try (OutputStream fileOut = new FileOutputStream("workbook.xls")) {
    wb.write(fileOut);
}

```

Creating Cells

```java

Workbook wb = new HSSFWorkbook();
//Workbook wb = new XSSFWorkbook();
```

```java
CreationHelper createHelper = wb.getCreationHelper();
Sheet sheet = wb.createSheet("new sheet");

// Create a row and put some cells in it. Rows are 0 based.
Row row = sheet.createRow(0);
// Create a cell and put a value in it.
Cell cell = row.createCell(0);
cell.setCellValue(1);

// Or do it on one line.
row.createCell(1).setCellValue(1.2);
row.createCell(2).setCellValue(
     createHelper.createRichTextString("This is a string"));
row.createCell(3).setCellValue(true);

// Write the output to a file
try (OutputStream fileOut = new FileOutputStream("workbook.xls")) {
    wb.write(fileOut);
}

```

Creating Date Cells

```java

Workbook wb = new HSSFWorkbook();
//Workbook wb = new XSSFWorkbook();
CreationHelper createHelper = wb.getCreationHelper();
Sheet sheet = wb.createSheet("new sheet");

// Create a row and put some cells in it. Rows are 0 based.
Row row = sheet.createRow(0);

```

```
// Create a cell and put a date value in it.  The first cell is not styled
// as a date.
Cell cell = row.createCell(0);
cell.setCellValue(new Date());

// we style the second cell as a date (and time).  It is important to
// create a new cell style from the workbook otherwise you can end up
// modifying the built in style and effecting not only this cell but other cells.
CellStyle cellStyle = wb.createCellStyle();
cellStyle.setDataFormat(
    createHelper.createDataFormat().getFormat("m/d/yy h:mm"));
cell = row.createCell(1);
cell.setCellValue(new Date());
cell.setCellStyle(cellStyle);

//you can also set date as java.util.Calendar
cell = row.createCell(2);
cell.setCellValue(Calendar.getInstance());
cell.setCellStyle(cellStyle);

// Write the output to a file
try (OutputStream fileOut = new FileOutputStream("workbook.xls")) {
    wb.write(fileOut);
}

```

Working with different types of cells

```

Workbook wb = new HSSFWorkbook();
Sheet sheet = wb.createSheet("new sheet");
```

```
Row row = sheet.createRow(2);
row.createCell(0).setCellValue(1.1);
row.createCell(1).setCellValue(new Date());
row.createCell(2).setCellValue(Calendar.getInstance());
row.createCell(3).setCellValue("a string");
row.createCell(4).setCellValue(true);
row.createCell(5).setCellType(CellType.ERROR);

// Write the output to a file
try (OutputStream fileOut = new FileOutputStream("workbook.xls")) {
    wb.write(fileOut);
}

```

Files vs InputStreams

When opening a workbook, either a .xls HSSFWorkbook, or a .xlsx XSSFWorkbook, the Workbook can be loaded fro either a *File* or an *InputStream*. Using a *File* object allows for lower memory consumption, while an *InputStream* requires more memory as it has to buffer the whole file.

If using *WorkbookFactory*, it's very easy to use one or the other:

```

// Use a file
Workbook wb = WorkbookFactory.create(new File("MyExcel.xls"));

// Use an InputStream, needs more memory
Workbook wb = WorkbookFactory.create(new FileInputStream("MyExcel.xlsx"));

```

If using *HSSFWorkbook* or *XSSFWorkbook* directly, you should generally go through *POIFSFileSystem* or *OPCPackage*, to have full control of the lifecycle (including closing the file when done):

```

// HSSFWorkbook, File
```

```
POIFSFileSystem fs = new POIFSFileSystem(new File("file.xls"));
HSSFWorkbook wb = new HSSFWorkbook(fs.getRoot(), true);
....
fs.close();

// HSSFWorkbook, InputStream, needs more memory
POIFSFileSystem fs = new POIFSFileSystem(myInputStream);
HSSFWorkbook wb = new HSSFWorkbook(fs.getRoot(), true);

// XSSFWorkbook, File
OPCPackage pkg = OPCPackage.open(new File("file.xlsx"));
XSSFWorkbook wb = new XSSFWorkbook(pkg);
....
pkg.close();

// XSSFWorkbook, InputStream, needs more memory
OPCPackage pkg = OPCPackage.open(myInputStream);
XSSFWorkbook wb = new XSSFWorkbook(pkg);
....
pkg.close();

```

Demonstrates various alignment options

```

public static void main(String[] args) throws Exception {
    Workbook wb = new XSSFWorkbook(); //or new HSSFWorkbook();

    Sheet sheet = wb.createSheet();
    Row row = sheet.createRow(2);
    row.setHeightInPoints(30);

    createCell(wb, row, 0, HorizontalAlignment.CENTER, VerticalAlignment.BOTTOM);
    createCell(wb, row, 1, HorizontalAlignment.CENTER_SELECTION, VerticalAlignment.BOTTOM);
    createCell(wb, row, 2, HorizontalAlignment.FILL, VerticalAlignment.CENTER);
    createCell(wb, row, 3, HorizontalAlignment.GENERAL, VerticalAlignment.CENTER);
    createCell(wb, row, 4, HorizontalAlignment.JUSTIFY, VerticalAlignment.JUSTIFY);
    createCell(wb, row, 5, HorizontalAlignment.LEFT, VerticalAlignment.TOP);
    createCell(wb, row, 6, HorizontalAlignment.RIGHT, VerticalAlignment.TOP);

    // Write the output to a file
    try (OutputStream fileOut = new FileOutputStream("xssf-align.xlsx")) {
        wb.write(fileOut);
    }

    wb.close();
}

/**
 * Creates a cell and aligns it a certain way.
 *
 * @param wb     the workbook
 * @param row    the row to create the cell in
 * @param column the column number to create the cell in
 * @param halign the horizontal alignment for the cell.
 * @param valign the vertical alignment for the cell.
 */
```

```
private static void createCell(Workbook wb, Row row, int column, HorizontalAlignment halign, VerticalAlignment valign) {
    Cell cell = row.createCell(column);
    cell.setCellValue("Align It");
    CellStyle cellStyle = wb.createCellStyle();
    cellStyle.setAlignment(halign);
    cellStyle.setVerticalAlignment(valign);
    cell.setCellStyle(cellStyle);
}

```

Working with borders

```

Workbook wb = new HSSFWorkbook();
Sheet sheet = wb.createSheet("new sheet");

// Create a row and put some cells in it. Rows are 0 based.
Row row = sheet.createRow(1);

// Create a cell and put a value in it.
Cell cell = row.createCell(1);
cell.setCellValue(4);

// Style the cell with borders all around.
CellStyle style = wb.createCellStyle();
style.setBorderBottom(BorderStyle.THIN);
style.setBottomBorderColor(IndexedColors.BLACK.getIndex());
style.setBorderLeft(BorderStyle.THIN);
style.setLeftBorderColor(IndexedColors.GREEN.getIndex());
style.setBorderRight(BorderStyle.THIN);
style.setRightBorderColor(IndexedColors.BLUE.getIndex());
```

```java
style.setBorderTop(BorderStyle.MEDIUM_DASHED);
style.setTopBorderColor(IndexedColors.BLACK.getIndex());
cell.setCellStyle(style);

// Write the output to a file
try (OutputStream fileOut = new FileOutputStream("workbook.xls")) {
    wb.write(fileOut);
}

wb.close();

```

Iterate over rows and cells

Sometimes, you'd like to just iterate over all the sheets in a workbook, all the rows in a sheet, or all the cells in a row. This is possible with a simple for loop.

These iterators are available by calling *workbook.sheetIterator()*, *sheet.rowIterator()*, and *row.cellIterator()*, or implicitly using a for-each loop. Note that a rowIterator and cellIterator iterate over rows or cells that have been created, skipping empty rows and cells.

```java

for (Sheet sheet : wb ) {
    for (Row row : sheet) {
        for (Cell cell : row) {
            // Do something here
        }
    }
}

```

Iterate over cells, with control of missing / blank cells

In some cases, when iterating, you need full control over how missing or blank rows and cells are treated, and you ne to ensure you visit every cell and not just those defined in the file. (The CellIterator will only return the cells defined i the file, which is largely those with values or stylings, but it depends on Excel).

In cases such as these, you should fetch the first and last column information for a row, then call *getCell(int, MissingCellPolicy)* to fetch the cell. Use a MissingCellPolicy to control how blank or null cells are handled.

```

// Decide which rows to process
int rowStart = Math.min(15, sheet.getFirstRowNum());
int rowEnd = Math.max(1400, sheet.getLastRowNum());

for (int rowNum = rowStart; rowNum < rowEnd; rowNum++) {
   Row r = sheet.getRow(rowNum);
   if (r == null) {
      // This whole row is empty
      // Handle it as needed
      continue;
   }

   int lastColumn = Math.max(r.getLastCellNum(), MY_MINIMUM_COLUMN_COUNT);

   for (int cn = 0; cn < lastColumn; cn++) {
      Cell c = r.getCell(cn, Row.RETURN_BLANK_AS_NULL);
      if (c == null) {
         // The spreadsheet is empty in this cell
      } else {
         // Do something useful with the cell's contents
      }
   }
}

```

Getting the cell contents

To get the contents of a cell, you first need to know what kind of cell it is (asking a string cell for its numeric contents will get you a NumberFormatException for example). So, you will want to switch on the cell's type, and then call the appropriate getter for that cell.

In the code below, we loop over every cell in one sheet, print out the cell's reference (eg A3), and then the cell's contents.

```

// import org.apache.poi.ss.usermodel.*;

DataFormatter formatter = new DataFormatter();
Sheet sheet1 = wb.getSheetAt(0);
for (Row row : sheet1) {
    for (Cell cell : row) {
        CellReference cellRef = new CellReference(row.getRowNum(), cell.getColumnIndex());
        System.out.print(cellRef.formatAsString());
        System.out.print(" - ");

        // get the text that appears in the cell by getting the cell value and applying any data formats (Date, 0.00, 1.23e9, $1.23, etc)
        String text = formatter.formatCellValue(cell);
        System.out.println(text);

        // Alternatively, get the value and format it yourself
        switch (cell.getCellType()) {
            case CellType.STRING:
                System.out.println(cell.getRichStringCellValue().getString());
                break;
            case CellType.NUMERIC:
                if (DateUtil.isCellDateFormatted(cell)) {
                    System.out.println(cell.getDateCellValue());
                } else {
                        System.out.println(cell.getNumericCellValue());
                    }
                    break;
                case CellType.BOOLEAN:
                    System.out.println(cell.getBooleanCellValue());
                    break;
                case CellType.FORMULA:
                    System.out.println(cell.getCellFormula());
                    break;
                case CellType.BLANK:
                    System.out.println();
                    break;
                default:
                    System.out.println();
            }
        }
}

```

Text Extraction

For most text extraction requirements, the standard ExcelExtractor class should provide all you need.

```

try (InputStream inp = new FileInputStream("workbook.xls")) {
    HSSFWorkbook wb = new HSSFWorkbook(new POIFSFileSystem(inp));
    ExcelExtractor extractor = new ExcelExtractor(wb);

    extractor.setFormulasNotResults(true);
    extractor.setIncludeSheetNames(false);
    String text = extractor.getText();
    wb.close();
}

```

For very fancy text extraction, XLS to CSV etc, take a look at */poi-examples/src/main/java/org/apache/poi/examples/hssf/eventusermodel/XLS2CSVmra.java*

Fills and colors

```

Workbook wb = new XSSFWorkbook();
Sheet sheet = wb.createSheet("new sheet");

// Create a row and put some cells in it. Rows are 0 based.
Row row = sheet.createRow(1);

// Aqua background
CellStyle style = wb.createCellStyle();
style.setFillBackgroundColor(IndexedColors.AQUA.getIndex());
style.setFillPattern(FillPatternType.BIG_SPOTS);
Cell cell = row.createCell(1);
cell.setCellValue("X");
cell.setCellStyle(style);

// Orange "foreground", foreground being the fill foreground not the font color.
style = wb.createCellStyle();
style.setFillForegroundColor(IndexedColors.ORANGE.getIndex());
style.setFillPattern(FillPatternType.SOLID_FOREGROUND);
cell = row.createCell(2);
cell.setCellValue("X");
cell.setCellStyle(style);

// Write the output to a file
```

```
try (OutputStream fileOut = new FileOutputStream("workbook.xls")) {
    wb.write(fileOut);
}

wb.close();

```

Merging cells

```

Workbook wb = new HSSFWorkbook();
Sheet sheet = wb.createSheet("new sheet");

Row row = sheet.createRow(1);
Cell cell = row.createCell(1);
cell.setCellValue("This is a test of merging");

sheet.addMergedRegion(new CellRangeAddress(
        1, //first row (0-based)
        1, //last row  (0-based)
        1, //first column (0-based)
        2  //last column  (0-based)
));

// Write the output to a file
try (OutputStream fileOut = new FileOutputStream("workbook.xls")) {
    wb.write(fileOut);
}

wb.close();

```

Working with fonts

```

Workbook wb = new HSSFWorkbook();
Sheet sheet = wb.createSheet("new sheet");

// Create a row and put some cells in it. Rows are 0 based.
Row row = sheet.createRow(1);

// Create a new font and alter it.
Font font = wb.createFont();
font.setFontHeightInPoints((short)24);
font.setFontName("Courier New");
font.setItalic(true);
font.setStrikeout(true);

// Fonts are set into a style so create a new one to use.
CellStyle style = wb.createCellStyle();
style.setFont(font);

// Create a cell and put a value in it.
Cell cell = row.createCell(1);
cell.setCellValue("This is a test of fonts");
cell.setCellStyle(style);

// Write the output to a file
try (OutputStream fileOut = new FileOutputStream("workbook.xls")) {
    wb.write(fileOut);
}

wb.close();

```

Note, the maximum number of unique fonts in a workbook is limited to 32767. You should re-use fonts in your applications instead of creating a font for each cell. Examples:

Wrong:

```

for (int i = 0; i < 10000; i++) {
    Row row = sheet.createRow(i);
    Cell cell = row.createCell(0);

    CellStyle style = workbook.createCellStyle();
    Font font = workbook.createFont();
    font.setBoldweight(Font.BOLDWEIGHT_BOLD);
    style.setFont(font);
    cell.setCellStyle(style);
}
```

Correct:

```

CellStyle style = workbook.createCellStyle();
Font font = workbook.createFont();
font.setBoldweight(Font.BOLDWEIGHT_BOLD);
style.setFont(font);
for (int i = 0; i < 10000; i++) {
    Row row = sheet.createRow(i);
    Cell cell = row.createCell(0);
    cell.setCellStyle(style);
}
```

Custom colors

HSSF:

```

```

```
HSSFWorkbook wb = new HSSFWorkbook();
HSSFSheet sheet = wb.createSheet();
HSSFRow row = sheet.createRow(0);
HSSFCell cell = row.createCell(0);
cell.setCellValue("Default Palette");

//apply some colors from the standard palette,
// as in the previous examples.
//we'll use red text on a lime background

HSSFCellStyle style = wb.createCellStyle();
style.setFillForegroundColor(HSSFColor.LIME.index);
style.setFillPattern(FillPatternType.SOLID_FOREGROUND);

HSSFFont font = wb.createFont();
font.setColor(HSSFColor.RED.index);
style.setFont(font);

cell.setCellStyle(style);

//save with the default palette
try (OutputStream out = new FileOutputStream("default_palette.xls")) {
    wb.write(out);
}

//now, let's replace RED and LIME in the palette
// with a more attractive combination
// (lovingly borrowed from freebsd.org)

cell.setCellValue("Modified Palette");

```

```
//creating a custom palette for the workbook
HSSFPalette palette = wb.getCustomPalette();

//replacing the standard red with freebsd.org red
palette.setColorAtIndex(HSSFColor.RED.index,
        (byte) 153,  //RGB red (0-255)
        (byte) 0,    //RGB green
        (byte) 0     //RGB blue
);
//replacing lime with freebsd.org gold
palette.setColorAtIndex(HSSFColor.LIME.index, (byte) 255, (byte) 204, (byte) 102);

//save with the modified palette
// note that wherever we have previously used RED or LIME, the
// new colors magically appear
try (out = new FileOutputStream("modified_palette.xls")) {
    wb.write(out);
}

```

XSSF:

```

XSSFWorkbook wb = new XSSFWorkbook();
XSSFSheet sheet = wb.createSheet();
XSSFRow row = sheet.createRow(0);
XSSFCell cell = row.createCell( 0);
cell.setCellValue("custom XSSF colors");

XSSFCellStyle style1 = wb.createCellStyle();
```

```
style1.setFillForegroundColor(new XSSFColor(new java.awt.Color(128, 0, 128),
new DefaultIndexedColorMap()));
style1.setFillPattern(FillPatternType.SOLID_FOREGROUND);

```

Reading and Rewriting Workbooks

```

try (InputStream inp = new FileInputStream("workbook.xls")) {
//InputStream inp = new FileInputStream("workbook.xlsx");

    Workbook wb = WorkbookFactory.create(inp);
    Sheet sheet = wb.getSheetAt(0);
    Row row = sheet.getRow(2);
    Cell cell = row.getCell(3);
    if (cell == null)
        cell = row.createCell(3);
    cell.setCellType(CellType.STRING);
    cell.setCellValue("a test");

    // Write the output to a file
    try (OutputStream fileOut = new FileOutputStream("workbook.xls")) {
        wb.write(fileOut);
    }
}

```

Using newlines in cells

```

Workbook wb = new XSSFWorkbook();   //or new HSSFWorkbook();
Sheet sheet = wb.createSheet();

```

```
Row row = sheet.createRow(2);
Cell cell = row.createCell(2);
cell.setCellValue("Use \n with word wrap on to create a new line");

//to enable newlines you need set a cell styles with wrap=true
CellStyle cs = wb.createCellStyle();
cs.setWrapText(true);
cell.setCellStyle(cs);

//increase row height to accommodate two lines of text
row.setHeightInPoints((2*sheet.getDefaultRowHeightInPoints()));

//adjust column width to fit the content
sheet.autoSizeColumn(2);

try (OutputStream fileOut = new FileOutputStream("ooxml-newlines.xlsx")) {
    wb.write(fileOut);
}

wb.close();

```

Data Formats

```

Workbook wb = new HSSFWorkbook();
Sheet sheet = wb.createSheet("format sheet");
CellStyle style;
DataFormat format = wb.createDataFormat();
Row row;
Cell cell;
int rowNum = 0;
int colNum = 0;

row = sheet.createRow(rowNum++);
cell = row.createCell(colNum);
cell.setCellValue(11111.25);
style = wb.createCellStyle();
style.setDataFormat(format.getFormat("0.0"));
cell.setCellStyle(style);

row = sheet.createRow(rowNum++);
cell = row.createCell(colNum);
cell.setCellValue(11111.25);
style = wb.createCellStyle();
style.setDataFormat(format.getFormat("#,##0.0000"));
cell.setCellStyle(style);

try (OutputStream fileOut = new FileOutputStream("workbook.xls")) {
    wb.write(fileOut);
}

wb.close();

```

Fit Sheet to One Page

```

Workbook wb = new HSSFWorkbook();
Sheet sheet = wb.createSheet("format sheet");
PrintSetup ps = sheet.getPrintSetup();

sheet.setAutobreaks(true);

```

```
ps.setFitHeight((short)1);
ps.setFitWidth((short)1);


// Create various cells and rows for spreadsheet.

try (OutputStream fileOut = new FileOutputStream("workbook.xls")) {
    wb.write(fileOut);
}

wb.close();

```

Set Print Area

```

Workbook wb = new HSSFWorkbook();
Sheet sheet = wb.createSheet("Sheet1");
//sets the print area for the first sheet
wb.setPrintArea(0, "$A$1:$C$2");

//Alternatively:
wb.setPrintArea(
        0, //sheet index
        0, //start column
        1, //end column
        0, //start row
        0  //end row
);

try (OutputStream fileOut = new FileOutputStream("workbook.xls")) {
    wb.write(fileOut);
}

wb.close();

```

Set Page Numbers on Footer

```

Workbook wb = new HSSFWorkbook(); // or new XSSFWorkbook();
Sheet sheet = wb.createSheet("format sheet");
Footer footer = sheet.getFooter();

footer.setRight( "Page " + HeaderFooter.page() + " of " +
HeaderFooter.numPages() );



// Create various cells and rows for spreadsheet.

try (OutputStream fileOut = new FileOutputStream("workbook.xls")) {
    wb.write(fileOut);
}

wb.close();

```

Using the Convenience Functions

The convenience functions provide utility features such as setting borders around merged regions and changing style attributes without explicitly creating new styles.

```

Workbook wb = new HSSFWorkbook();  // or new XSSFWorkbook()
Sheet sheet1 = wb.createSheet( "new sheet" );

```

```
// Create a merged region
Row row = sheet1.createRow( 1 );
Row row2 = sheet1.createRow( 2 );
Cell cell = row.createCell( 1 );
cell.setCellValue( "This is a test of merging" );
CellRangeAddress region = CellRangeAddress.valueOf("B2:E5");
sheet1.addMergedRegion( region );

// Set the border and border colors.
RegionUtil.setBorderBottom( BorderStyle.MEDIUM_DASHED, region, sheet1, wb );
RegionUtil.setBorderTop(    BorderStyle.MEDIUM_DASHED, region, sheet1, wb );
RegionUtil.setBorderLeft(   BorderStyle.MEDIUM_DASHED, region, sheet1, wb );
RegionUtil.setBorderRight(  BorderStyle.MEDIUM_DASHED, region, sheet1, wb );
RegionUtil.setBottomBorderColor(IndexedColors.AQUA.getIndex(), region,
sheet1, wb);
RegionUtil.setTopBorderColor(   IndexedColors.AQUA.getIndex(), region,
sheet1, wb);
RegionUtil.setLeftBorderColor(  IndexedColors.AQUA.getIndex(), region,
sheet1, wb);
RegionUtil.setRightBorderColor( IndexedColors.AQUA.getIndex(), region,
sheet1, wb);

// Shows some usages of HSSFCellUtil
CellStyle style = wb.createCellStyle();
style.setIndention((short)4);
CellUtil.createCell(row, 8, "This is the value of the cell", style);
Cell cell2 = CellUtil.createCell( row2, 8, "This is the value of the cell");
CellUtil.setAlignment(cell2, HorizontalAlignment.CENTER);

// Write out the workbook
try (OutputStream fileOut = new FileOutputStream( "workbook.xls" )) {
    wb.write( fileOut );
}

wb.close();

```

Shift rows up or down on a sheet

```

Workbook wb = new HSSFWorkbook();
Sheet sheet = wb.createSheet("row sheet");

// Create various cells and rows for spreadsheet.

// Shift rows 6 - 11 on the spreadsheet to the top (rows 0 - 5)
sheet.shiftRows(5, 10, -5);


```

Set a sheet as selected

```

Workbook wb = new HSSFWorkbook();
Sheet sheet = wb.createSheet("row sheet");
sheet.setSelected(true);


```

Set the zoom magnification

The zoom is expressed as a fraction. For example to express a zoom of 75% use 3 for the numerator and 4 for the denominator.

```

Workbook wb = new HSSFWorkbook();
Sheet sheet1 = wb.createSheet("new sheet");
```

```
sheet1.setZoom(75);   // 75 percent magnification

```

Splits and freeze panes

There are two types of panes you can create; freeze panes and split panes.

A freeze pane is split by columns and rows. You create a freeze pane using the following mechanism:

sheet1.createFreezePane(3, 2, 3, 2);

The first two parameters are the columns and rows you wish to split by. The second two parameters indicate the cells that are visible in the bottom right quadrant.

Split panes appear differently. The split area is divided into four separate work area's. The split occurs at the pixel lev and the user is able to adjust the split by dragging it to a new position.

Split panes are created with the following call:

sheet2.createSplitPane(2000, 2000, 0, 0, Sheet.PANE_LOWER_LEFT);

The first parameter is the x position of the split. This is in 1/20th of a point. A point in this case seems to equate to a pixel. The second parameter is the y position of the split. Again in 1/20th of a point.

The last parameter indicates which pane currently has the focus. This will be one of Sheet.PANE_LOWER_LEFT, PANE_LOWER_RIGHT, PANE_UPPER_RIGHT or PANE_UPPER_LEFT.

```

Workbook wb = new HSSFWorkbook();
Sheet sheet1 = wb.createSheet("new sheet");
Sheet sheet2 = wb.createSheet("second sheet");
Sheet sheet3 = wb.createSheet("third sheet");
Sheet sheet4 = wb.createSheet("fourth sheet");

// Freeze just one row
sheet1.createFreezePane( 0, 1, 0, 1 );
// Freeze just one column
sheet2.createFreezePane( 1, 0, 1, 0 );
```

```
// Freeze the columns and rows (forget about scrolling position of the lower right quadrant).
sheet3.createFreezePane( 2, 2 );
// Create a split with the lower left side being the active quadrant
sheet4.createSplitPane( 2000, 2000, 0, 0, Sheet.PANE_LOWER_LEFT );

try (OutputStream fileOut = new FileOutputStream("workbook.xls")) {
    wb.write(fileOut);
}

```

Repeating rows and columns

It's possible to set up repeating rows and columns in your printouts by using the setRepeatingRows() and setRepeatingColumns() methods in the Sheet class.

These methods expect a CellRangeAddress parameter which specifies the range for the rows or columns to repeat. Fo setRepeatingRows(), it should specify a range of rows to repeat, with the column part spanning all columns. For setRepeatingColumns(), it should specify a range of columns to repeat, with the row part spanning all rows. If the parameter is null, the repeating rows or columns will be removed.

```

Workbook wb = new HSSFWorkbook();           // or new XSSFWorkbook();
Sheet sheet1 = wb.createSheet("Sheet1");
Sheet sheet2 = wb.createSheet("Sheet2");

// Set the rows to repeat from row 4 to 5 on the first sheet.
sheet1.setRepeatingRows(CellRangeAddress.valueOf("4:5"));
// Set the columns to repeat from column A to C on the second sheet
sheet2.setRepeatingColumns(CellRangeAddress.valueOf("A:C"));

try (OutputStream fileOut = new FileOutputStream("workbook.xls")) {
    wb.write(fileOut);
}
```

Headers and Footers

Example is for headers but applies directly to footers.

```

Workbook wb = new HSSFWorkbook();
Sheet sheet = wb.createSheet("new sheet");

Header header = sheet.getHeader();
header.setCenter("Center Header");
header.setLeft("Left Header");
header.setRight(HSSFHeader.font("Stencil-Normal", "Italic") +
                HSSFHeader.fontSize((short) 16) + "Right w/ Stencil-Normal
Italic font and size 16");

try (OutputStream fileOut = new FileOutputStream("workbook.xls")) {
    wb.write(fileOut);
}

```

XSSF Enhancement for Headers and Footers

Example is for headers but applies directly to footers. Note, the above example for basic headers and footers applies t
XSSF Workbooks as well as HSSF Workbooks. The HSSFHeader stuff does not work for XSSF Workbooks.

XSSF has the ability to handle First page headers and footers, as well as Even/Odd headers and footers. All Header/Footer Property flags can be handled in XSSF as well. The odd header and footer is the default header and footer. It is displayed on all pages that do not display either a first page header or an even page header. That is, if the Even header/footer does not exist, then the odd header/footer is displayed on even pages. If the first page header/footer does not exist, then the odd header/footer is displayed on the first page. If the even/odd property is no
set, that is the same as the even header/footer not existing. If the first page property does not exist, that is the same a
the first page header/footer not existing.

```

Workbook wb = new XSSFWorkbook();
```

```
XSSFSheet sheet = (XSSFSheet) wb.createSheet("new sheet");

// Create a first page header
Header header = sheet.getFirstHeader();
header.setCenter("Center First Page Header");
header.setLeft("Left First Page Header");
header.setRight("Right First Page Header");

// Create an even page header
Header header2 = sheet.getEvenHeader();
der2.setCenter("Center Even Page Header");
header2.setLeft("Left Even Page Header");
header2.setRight("Right Even Page Header");

// Create an odd page header
Header header3 = sheet.getOddHeader();
der3.setCenter("Center Odd Page Header");
header3.setLeft("Left Odd Page Header");
header3.setRight("Right Odd Page Header");

// Set/Remove Header properties
XSSFHeaderProperties prop = sheet.getHeaderFooterProperties();
prop.setAlignWithMargins();
prop.scaleWithDoc();
prop.removeDifferentFirstPage(); // This does not remove first page headers or footers
prop.removeDifferentEvenOdd(); // This does not remove even headers or footers

try (OutputStream fileOut = new FileOutputStream("workbook.xlsx")) {
    wb.write(fileOut);
}
```

Drawing Shapes

POI supports drawing shapes using the Microsoft Office drawing tools. Shapes on a sheet are organized in a hierarch of groups and and shapes. The top-most shape is the patriarch. This is not visible on the sheet at all. To start drawing you need to call `createPatriarch` on the `HSSFSheet` class. This has the effect erasing any other shape information stored in that sheet. By default POI will leave shape records alone in the sheet unless you make a call to this method.

To create a shape you have to go through the following steps:

1. Create the patriarch.
2. Create an anchor to position the shape on the sheet.
3. Ask the patriarch to create the shape.
4. Set the shape type (line, oval, rectangle etc...)
5. Set any other style details concerning the shape. (eg: line thickness, etc...)

```

HSSFPatriarch patriarch = sheet.createDrawingPatriarch();
a = new HSSFClientAnchor( 0, 0, 1023, 255, (short) 1, 0, (short) 1, 0 );
HSSFSimpleShape shape1 = patriarch.createSimpleShape(a1);
shape1.setShapeType(HSSFSimpleShape.OBJECT_TYPE_LINE);

```

Text boxes are created using a different call:

```

HSSFTextbox textbox1 = patriarch.createTextbox(
        new HSSFClientAnchor(0,0,0,0,(short)1,1,(short)2,2));
textbox1.setString(new HSSFRichTextString("This is a test") );

```

It's possible to use different fonts to style parts of the text in the textbox. Here's how:

```

HSSFFont font = wb.createFont();
font.setItalic(true);
font.setUnderline(HSSFFont.U_DOUBLE);
HSSFRichTextString string = new HSSFRichTextString("Woo!!!");
string.applyFont(2,5,font);
textbox.setString(string );

```

Just as can be done manually using Excel, it is possible to group shapes together. This is done by calling `createGroup()` and then creating the shapes using those groups.

It's also possible to create groups within groups.

Warning

Any group you create should contain at least two other shapes or subgroups.

Here's how to create a shape group:

```

// Create a shape group.
HSSFShapeGroup group = patriarch.createGroup(
        new HSSFClientAnchor(0,0,900,200,(short)2,2,(short)2,2));

// Create a couple of lines in the group.
HSSFSimpleShape shape1 = group.createShape(new HSSFChildAnchor(3,3,500,500));
shape1.setShapeType(HSSFSimpleShape.OBJECT_TYPE_LINE);
( (HSSFChildAnchor) shape1.getAnchor() ).setAnchor(3,3,500,500);
HSSFSimpleShape shape2 = group.createShape(new HSSFChildAnchor(1,200,400,600));
shape2.setShapeType(HSSFSimpleShape.OBJECT_TYPE_LINE);

```

If you're being observant you'll noticed that the shapes that are added to the group use a new type of anchor: the `HSSFChildAnchor`. What happens is that the created group has its own coordinate space for shapes that are place into it. POI defaults this to (0,0,1023,255) but you are able to change it as desired. Here's how:

```

myGroup.setCoordinates(10,10,20,20); // top-left, bottom-right

```

If you create a group within a group it's also going to have its own coordinate space.

Styling Shapes

By default shapes can look a little plain. It's possible to apply different styles to the shapes however. The sorts of thin that can currently be done are:

- Change the fill color.
- Make a shape with no fill color.
- Change the thickness of the lines.
- Change the style of the lines. Eg: dashed, dotted.
- Change the line color.

Here's an examples of how this is done:

```

HSSFSimpleShape s = patriarch.createSimpleShape(a);
s.setShapeType(HSSFSimpleShape.OBJECT_TYPE_OVAL);
s.setLineStyleColor(10,10,10);
s.setFillColor(90,10,200);
s.setLineWidth(HSSFShape.LINEWIDTH_ONE_PT * 3);
s.setLineStyle(HSSFShape.LINESTYLE_DOTSYS);

```

Shapes and Graphics2d

While the native POI shape drawing commands are the recommended way to draw shapes in a shape it's sometimes desirable to use a standard API for compatibility with external libraries. With this in mind we created some wrappers for `Graphics` and `Graphics2d`.

Warning

It's important to not however before continuing that `Graphics2d` is a poor match to the capabilities of the Microsoft Office drawing commands. The older `Graphics` class offers a closer match but is still a square peg in a round hole.

All Graphics commands are issued into an `HSSFShapeGroup`. Here's how it's done:

```

a = new HSSFClientAnchor( 0, 0, 1023, 255, (short) 1, 0, (short) 1, 0 );
group = patriarch.createGroup( a );
group.setCoordinates( 0, 0, 80 * 4 , 12 * 23  );
float verticalPointsPerPixel = a.getAnchorHeightInPoints(sheet) /
(float)Math.abs(group.getY2() - group.getY1());
g = new EscherGraphics( group, wb, Color.black, verticalPointsPerPixel );
g2d = new EscherGraphics2d( g );
drawChemicalStructure( g2d );

```

The first thing we do is create the group and set its coordinates to match what we plan to draw. Next we calculate a reasonable fontSizeMultiplier then create the EscherGraphics object. Since what we really want is a `Graphics2d` object we create an EscherGraphics2d object and pass in the graphics object we created. Finally we call a routine that draws into the EscherGraphics2d object.

The vertical points per pixel deserves some more explanation. One of the difficulties in converting Graphics calls into escher drawing calls is that Excel does not have the concept of absolute pixel positions. It measures its cell widths in 'characters' and the cell heights in points. Unfortunately it's not defined exactly what type of character it's measuring Presumably this is due to the fact that the Excel will be using different fonts on different platforms or even within the same platform.

Because of this constraint we've had to implement the concept of a verticalPointsPerPixel. This the amount the font should be scaled by when you issue commands such as drawString(). To calculate this value use the follow formula:

```

multipler = groupHeightInPoints / heightOfGroup

```

The height of the group is calculated fairly simply by calculating the difference between the y coordinates of the bounding box of the shape. The height of the group can be calculated by using a convenience called `HSSFClientAnchor.getAnchorHeightInPoints()`.

Many of the functions supported by the graphics classes are not complete. Here's some of the functions that are know to work.

- fillRect()
- fillOval()
- drawString()
- drawOval()
- drawLine()
- clearRect()

Functions that are not supported will return and log a message using the POI logging infrastructure (disabled by default).

Outlining

Outlines are great for grouping sections of information together and can be added easily to columns and rows using t POI API. Here's how:

```

Workbook wb = new HSSFWorkbook();
Sheet sheet1 = wb.createSheet("new sheet");

sheet1.groupRow( 5, 14 );
sheet1.groupRow( 7, 14 );
sheet1.groupRow( 16, 19 );

sheet1.groupColumn( 4, 7 );
sheet1.groupColumn( 9, 12 );
sheet1.groupColumn( 10, 11 );

try (OutputStream fileOut = new FileOutputStream(filename)) {
    wb.write(fileOut);
}

```

To collapse (or expand) an outline use the following calls:

```

sheet1.setRowGroupCollapsed( 7, true );
sheet1.setColumnGroupCollapsed( 4, true );

```

The row/column you choose should contain an already created group. It can be anywhere within the group.

Images

Images are part of the drawing support. To add an image just call `createPicture()` on the drawing patriarch. At the time of writing the following types are supported:

- PNG
- JPG
- DIB

It should be noted that any existing drawings may be erased once you add an image to a sheet.

```

//create a new workbook
Workbook wb = new XSSFWorkbook(); //or new HSSFWorkbook();

//add picture data to this workbook.
InputStream is = new FileInputStream("image1.jpeg");
byte[] bytes = IOUtils.toByteArray(is);
int pictureIdx = wb.addPicture(bytes, Workbook.PICTURE_TYPE_JPEG);
is.close();

CreationHelper helper = wb.getCreationHelper();
```

```

//create sheet
Sheet sheet = wb.createSheet();

// Create the drawing patriarch.  This is the top level container for all shapes.
Drawing drawing = sheet.createDrawingPatriarch();

//add a picture shape
ClientAnchor anchor = helper.createClientAnchor();
//set top-left corner of the picture,
//subsequent call of Picture#resize() will operate relative to it
anchor.setCol1(3);
anchor.setRow1(2);
Picture pict = drawing.createPicture(anchor, pictureIdx);

//auto-size picture relative to its top-left corner
pict.resize();

//save workbook
String file = "picture.xls";
if(wb instanceof XSSFWorkbook) file += "x";
try (OutputStream fileOut = new FileOutputStream(file)) {
    wb.write(fileOut);
}

```

Warning

Picture.resize() works only for JPEG and PNG. Other formats are not yet supported.

Reading images from a workbook:

```


ist lst = workbook.getAllPictures();
or (Iterator it = lst.iterator(); it.hasNext(); ) {
   PictureData pict = (PictureData)it.next();
   String ext = pict.suggestFileExtension();
   byte[] data = pict.getData();
   if (ext.equals("jpeg")){
     try (OutputStream out = new FileOutputStream("pict.jpg")) {
       out.write(data);
     }
   }


```

Named Ranges and Named Cells

Named Range is a way to refer to a group of cells by a name. Named Cell is a degenerate case of Named Range in that the 'group of cells' contains exactly one cell. You can create as well as refer to cells in a workbook by their named rang When working with Named Ranges, the classes `org.apache.poi.ss.util.CellReference` and `org.apache.poi.ss.util.AreaReference` are used.

Note: Using relative values like 'A1:B1' can lead to unexpected moving of the cell that the name points to when workin with the workbook in Microsoft Excel, usually using absolute references like '$A$1:$B$1' avoids this, see also this discussion.

Creating Named Range / Named Cell

```

// setup code
String sname = "TestSheet", cname = "TestName", cvalue = "TestVal";
Workbook wb = new HSSFWorkbook();
Sheet sheet = wb.createSheet(sname);
sheet.createRow(0).createCell(0).setCellValue(cvalue);
```

```

// 1. create named range for a single cell using areareference
Name namedCell = wb.createName();
namedCell.setNameName(cname + "1");
String reference = sname+"!$A$1:$A$1"; // area reference
namedCell.setRefersToFormula(reference);

// 2. create named range for a single cell using cellreference
Name namedCel2 = wb.createName();
namedCel2.setNameName(cname + "2");
reference = sname+"!$A$1"; // cell reference
namedCel2.setRefersToFormula(reference);

// 3. create named range for an area using AreaReference
Name namedCel3 = wb.createName();
namedCel3.setNameName(cname + "3");
reference = sname+"!$A$1:$C$5"; // area reference
namedCel3.setRefersToFormula(reference);

// 4. create named formula
Name namedCel4 = wb.createName();
namedCel4.setNameName("my_sum");
namedCel4.setRefersToFormula("SUM(" + sname + "!$I$2:$I$6)");

```

Reading from Named Range / Named Cell

```

// setup code
String cname = "TestName";
Workbook wb = getMyWorkbook(); // retrieve workbook

```

```
// retrieve the named range
int namedCellIdx = wb.getNameIndex(cellName);
Name aNamedCell = wb.getNameAt(namedCellIdx);

// retrieve the cell at the named range and test its contents
AreaReference aref = new AreaReference(aNamedCell.getRefersToFormula());
CellReference[] crefs = aref.getAllReferencedCells();
for (int i=0; i<crefs.length; i++) {
    Sheet s = wb.getSheet(crefs[i].getSheetName());
    Row r = sheet.getRow(crefs[i].getRow());
    Cell c = r.getCell(crefs[i].getCol());
    // extract the cell contents based on cell type etc.
}

```

Reading from non-contiguous Named Ranges

```

// Setup code
String cname = "TestName";
Workbook wb = getMyWorkbook(); // retrieve workbook

// Retrieve the named range
// Will be something like "$C$10,$D$12:$D$14";
int namedCellIdx = wb.getNameIndex(cellName);
Name aNamedCell = wb.getNameAt(namedCellIdx);

// Retrieve the cell at the named range and test its contents
// Will get back one AreaReference for C10, and
//  another for D12 to D14
AreaReference[] arefs =
AreaReference.generateContiguous(aNamedCell.getRefersToFormula());
```

```
for (int i=0; i<arefs.length; i++) {
    // Only get the corners of the Area
    // (use arefs[i].getAllReferencedCells() to get all cells)
    CellReference[] crefs = arefs[i].getCells();
    for (int j=0; j<crefs.length; j++) {
        // Check it turns into real stuff
        Sheet s = wb.getSheet(crefs[j].getSheetName());
        Row r = s.getRow(crefs[j].getRow());
        Cell c = r.getCell(crefs[j].getCol());
        // Do something with this corner cell
    }
}

```

Note, when a cell is deleted, Excel does not delete the attached named range. As result, workbook can contain named ranges that point to cells that no longer exist. You should check the validity of a reference before constructing AreaReference

```

if(name.isDeleted()){
  //named range points to a deleted cell.
} else {
  AreaReference ref = new AreaReference(name.getRefersToFormula());
}

```

Cell Comments - HSSF and XSSF

A comment is a rich text note that is attached to & associated with a cell, separate from other cell content. Comment content is stored separate from the cell, and is displayed in a drawing object (like a text box) that is separate from, bu associated with, a cell

```

```

```
Workbook wb = new XSSFWorkbook(); //or new HSSFWorkbook();

CreationHelper factory = wb.getCreationHelper();

Sheet sheet = wb.createSheet();

Row row   = sheet.createRow(3);
Cell cell = row.createCell(5);
cell.setCellValue("F4");

Drawing drawing = sheet.createDrawingPatriarch();

// When the comment box is visible, have it show in a 1x3 space
ClientAnchor anchor = factory.createClientAnchor();
anchor.setCol1(cell.getColumnIndex());
anchor.setCol2(cell.getColumnIndex()+1);
anchor.setRow1(row.getRowNum());
anchor.setRow2(row.getRowNum()+3);

// Create the comment and set the text+author
Comment comment = drawing.createCellComment(anchor);
RichTextString str = factory.createRichTextString("Hello, World!");
comment.setString(str);
comment.setAuthor("Apache POI");

// Assign the comment to the cell
cell.setCellComment(comment);

String fname = "comment-xssf.xls";
if(wb instanceof XSSFWorkbook) fname += "x";
try (OutputStream out = new FileOutputStream(fname)) {
    wb.write(out);
}

wb.close();

```

Reading cell comments

```

Cell cell = sheet.get(3).getColumn(1);
Comment comment = cell.getCellComment();
if (comment != null) {
  RichTextString str = comment.getString();
  String author = comment.getAuthor();
}
//  alternatively you can retrieve cell comments by (row, column)
comment = sheet.getCellComment(3, 1);

```

To get all the comments on a sheet:

```

Map<CellAddress, Comment> comments = sheet.getCellComments();
Comment commentA1 = comments.get(new CellAddress(0, 0));
Comment commentB1 = comments.get(new CellAddress(0, 1));
for (Entry<CellAddress, ? extends Comment> e : comments.entrySet()) {
  CellAddress loc = e.getKey();
  Comment comment = e.getValue();
  System.out.println("Comment at " + loc + ": " +
      "[" + comment.getAuthor() + "] " + comment.getString().getString());
}

```

Adjust column width to fit the contents

```java

Sheet sheet = workbook.getSheetAt(0);
sheet.autoSizeColumn(0); //adjust width of the first column
sheet.autoSizeColumn(1); //adjust width of the second column

```

For SXSSFWorkbooks only, because the random access window is likely to exclude most of the rows in the worksheet which are needed for computing the best-fit width of a column, the columns must be tracked for auto-sizing prior to flushing any rows.

```java

SXSSFWorkbook workbook = new SXSSFWorkbook();
SXSSFSheet sheet = workbook.createSheet();
sheet.trackColumnForAutoSizing(0);
sheet.trackColumnForAutoSizing(1);
// If you have a Collection of column indices, see SXSSFSheet#trackColumnForAutoSizing(Collection<Integer>)
// or roll your own for-loop.
// Alternatively, use SXSSFSheet#trackAllColumnsForAutoSizing() if the columns that will be auto-sized aren't
// known in advance or you are upgrading existing code and are trying to minimize changes. Keep in mind
// that tracking all columns will require more memory and CPU cycles, as the best-fit width is calculated
// on all tracked columns on every row that is flushed.

// create some cells
for (int r=0; r < 10; r++) {
    Row row = sheet.createRow(r);
    for (int c; c < 10; c++) {
        Cell cell = row.createCell(c);
```

```
            cell.setCellValue("Cell " + c.getAddress().formatAsString());
        }
}

// Auto-size the columns.
sheet.autoSizeColumn(0);
sheet.autoSizeColumn(1);

```

Note, that Sheet#autoSizeColumn() does not evaluate formula cells, the width of formula cells is calculated based on the cached formula result. If your workbook has many formulas then it is a good idea to evaluate them before auto-sizing.

Warning

To calculate column width Sheet.autoSizeColumn uses Java2D classes that throw exception if graphical environment is not available. In case if graphical environment is not available, you must tell Java that you are running in headless mode and set the following system property: `java.awt.headless=true` . You should also ensure that the fonts you use in your workbook are available to Java.

How to read hyperlinks

```

Sheet sheet = workbook.getSheetAt(0);

Cell cell = sheet.getRow(0).getCell(0);
Hyperlink link = cell.getHyperlink();
if(link != null){
    System.out.println(link.getAddress());
}

```

How to create hyperlinks

```

```

```
Workbook wb = new XSSFWorkbook(); //or new HSSFWorkbook();
CreationHelper createHelper = wb.getCreationHelper();

//cell style for hyperlinks
//by default hyperlinks are blue and underlined
CellStyle hlink_style = wb.createCellStyle();
Font hlink_font = wb.createFont();
hlink_font.setUnderline(Font.U_SINGLE);
hlink_font.setColor(IndexedColors.BLUE.getIndex());
hlink_style.setFont(hlink_font);

Cell cell;
Sheet sheet = wb.createSheet("Hyperlinks");
//URL
cell = sheet.createRow(0).createCell(0);
cell.setCellValue("URL Link");

Hyperlink link = createHelper.createHyperlink(HyperlinkType.URL);
link.setAddress("https://poi.apache.org/");
cell.setHyperlink(link);
cell.setCellStyle(hlink_style);

//link to a file in the current directory
cell = sheet.createRow(1).createCell(0);
cell.setCellValue("File Link");
link = createHelper.createHyperlink(HyperlinkType.FILE);
link.setAddress("link1.xls");
cell.setHyperlink(link);
cell.setCellStyle(hlink_style);

```

```
//e-mail link
cell = sheet.createRow(2).createCell(0);
cell.setCellValue("Email Link");
link = createHelper.createHyperlink(HyperlinkType.EMAIL);
//note, if subject contains white spaces, make sure they are url-encoded
link.setAddress("mailto:poi@apache.org?subject=Hyperlinks");
cell.setHyperlink(link);
cell.setCellStyle(hlink_style);

//link to a place in this workbook

//create a target sheet and cell
Sheet sheet2 = wb.createSheet("Target Sheet");
sheet2.createRow(0).createCell(0).setCellValue("Target Cell");

cell = sheet.createRow(3).createCell(0);
cell.setCellValue("Worksheet Link");
Hyperlink link2 = createHelper.createHyperlink(HyperlinkType.DOCUMENT);
link2.setAddress("'Target Sheet'!A1");
cell.setHyperlink(link2);
cell.setCellStyle(hlink_style);

try (OutputStream out = new FileOutputStream("hyperinks.xlsx")) {
    wb.write(out);
}

wb.close();

```

Data Validations

As of version 3.8, POI has slightly different syntax to work with data validations with .xls and .xlsx formats.

hssf.usermodel (binary .xls format)

Check the value a user enters into a cell against one or more predefined value(s).

The following code will limit the value the user can enter into cell A1 to one of three integer values, 10, 20 or 30.

```

HSSFWorkbook workbook = new HSSFWorkbook();
HSSFSheet sheet = workbook.createSheet("Data Validation");
CellRangeAddressList addressList = new CellRangeAddressList(
  0, 0, 0, 0);
DVConstraint dvConstraint = DVConstraint.createExplicitListConstraint(
  new String[]{"10", "20", "30"});
DataValidation dataValidation = new HSSFDataValidation
  (addressList, dvConstraint);
dataValidation.setSuppressDropDownArrow(true);
sheet.addValidationData(dataValidation);

```

Drop Down Lists:

This code will do the same but offer the user a drop down list to select a value from.

```

HSSFWorkbook workbook = new HSSFWorkbook();
HSSFSheet sheet = workbook.createSheet("Data Validation");
CellRangeAddressList addressList = new CellRangeAddressList(
  0, 0, 0, 0);
DVConstraint dvConstraint = DVConstraint.createExplicitListConstraint(
  new String[]{"10", "20", "30"});
DataValidation dataValidation = new HSSFDataValidation
  (addressList, dvConstraint);
```

```
dataValidation.setSuppressDropDownArrow(false);
sheet.addValidationData(dataValidation);

```

Messages On Error:

To create a message box that will be shown to the user if the value they enter is invalid.

```

dataValidation.setErrorStyle(DataValidation.ErrorStyle.STOP);
dataValidation.createErrorBox("Box Title", "Message Text");

```

Replace 'Box Title' with the text you wish to display in the message box's title bar and 'Message Text' with the text of your error message.

Prompts:

To create a prompt that the user will see when the cell containing the data validation receives focus

```

dataValidation.createPromptBox("Title", "Message Text");
dataValidation.setShowPromptBox(true);

```

The text encapsulated in the first parameter passed to the createPromptBox() method will appear emboldened and a title to the prompt whilst the second will be displayed as the text of the message. The createExplicitListConstraint() method can be passed and array of String(s) containing interger, floating point, dates or text values.

Further Data Validations:

To obtain a validation that would check the value entered was, for example, an integer between 10 and 100, use the DVConstraint.createNumericConstraint(int, int, String, String) factory method.

```

dvConstraint = DVConstraint.createNumericConstraint(
  DVConstraint.ValidationType.INTEGER,
  DVConstraint.OperatorType.BETWEEN, "10", "100");
```

Look at the javadoc for the other validation and operator types; also note that not all validation types are supported f
this method. The values passed to the two String parameters can be formulas; the '=' symbol is used to denote a form

```

dvConstraint = DVConstraint.createNumericConstraint(
  DVConstraint.ValidationType.INTEGER,
  DVConstraint.OperatorType.BETWEEN, "=SUM(A1:A3)", "100");

```

It is not possible to create a drop down list if the createNumericConstraint() method is called, the setSuppressDropDownArrow(false) method call will simply be ignored.

Date and time constraints can be created by calling the createDateConstraint(int, String, String, String) or the createTimeConstraint(int, String, String). Both are very similar to the above and are explained in the javadoc.

Creating Data Validations From Spreadsheet Cells.

The contents of specific cells can be used to provide the values for the data validation and the DVConstraint.createFormulaListConstraint(String) method supports this. To specify that the values come from a contiguous range of cells do either of the following:

```

dvConstraint = DVConstraint.createFormulaListConstraint("$A$1:$A$3");

```

or

```

Name namedRange = workbook.createName();
namedRange.setNameName("list1");
namedRange.setRefersToFormula("$A$1:$A$3");
dvConstraint = DVConstraint.createFormulaListConstraint("list1");

```

and in both cases the user will be able to select from a drop down list containing the values from cells A1, A2 and A3.

The data does not have to be as the data validation. To select the data from a different sheet however, the sheet must given a name when created and that name should be used in the formula. So assuming the existence of a sheet name 'Data Sheet' this will work:

```

Name namedRange = workbook.createName();
namedRange.setNameName("list1");
namedRange.setRefersToFormula("'Data Sheet'!$A$1:$A$3");
dvConstraint = DVConstraint.createFormulaListConstraint("list1");

```

as will this:

```

dvConstraint = DVConstraint.createFormulaListConstraint("'Data
Sheet'!$A$1:$A$3");

```

whilst this will not:

```

Name namedRange = workbook.createName();
namedRange.setNameName("list1");
namedRange.setRefersToFormula("'Sheet1'!$A$1:$A$3");
dvConstraint = DVConstraint.createFormulaListConstraint("list1");

```

and nor will this:

```

dvConstraint =
DVConstraint.createFormulaListConstraint("'Sheet1'!$A$1:$A$3");

```

xssf.usermodel (.xlsx format)

Data validations work similarly when you are creating an xml based, SpreadsheetML, workbook file; but there are differences. Explicit casts are required, for example, in a few places as much of the support for data validations in the xssf stream was built into the unifying ss stream, of which more later. Other differences are noted with comments in t code.

Check the value the user enters into a cell against one or more predefined value(s).

```

XSSFWorkbook workbook = new XSSFWorkbook();
XSSFSheet sheet = workbook.createSheet("Data Validation");
XSSFDataValidationHelper dvHelper = new XSSFDataValidationHelper(sheet);
XSSFDataValidationConstraint dvConstraint = (XSSFDataValidationConstraint)
  dvHelper.createExplicitListConstraint(new String[]{"11", "21", "31"});
CellRangeAddressList addressList = new CellRangeAddressList(0, 0, 0, 0);
XSSFDataValidation validation = (XSSFDataValidation)dvHelper.createValidation(
  dvConstraint, addressList);

// Here the boolean value false is passed to the setSuppressDropDownArrow()
// method. In the hssf.usermodel examples above, the value passed to this
// method is true.
validation.setSuppressDropDownArrow(false);

// Note this extra method call. If this method call is omitted, or if the
// boolean value false is passed, then Excel will not validate the value the
// user enters into the cell.
validation.setShowErrorBox(true);
sheet.addValidationData(validation);
```

Drop Down Lists:

This code will do the same but offer the user a drop down list to select a value from.

```

```

```
XSSFWorkbook workbook = new XSSFWorkbook();
XSSFSheet sheet = workbook.createSheet("Data Validation");
XSSFDataValidationHelper dvHelper = new XSSFDataValidationHelper(sheet);
XSSFDataValidationConstraint dvConstraint = (XSSFDataValidationConstraint)
  dvHelper.createExplicitListConstraint(new String[]{"11", "21", "31"});
CellRangeAddressList addressList = new CellRangeAddressList(0, 0, 0, 0);
XSSFDataValidation validation =
(XSSFDataValidation)dvHelper.createValidation(
  dvConstraint, addressList);
validation.setShowErrorBox(true);
sheet.addValidationData(validation);
```

Note that the call to the setSuppressDropDowmArrow() method can either be simply excluded or replaced with:

```

validation.setSuppressDropDownArrow(true);
```

Prompts and Error Messages:

These both exactly mirror the hssf.usermodel so please refer to the 'Messages On Error:' and 'Prompts:' sections abov

Further Data Validations:

To obtain a validation that would check the value entered was, for example, an integer between 10 and 100, use the XSSFDataValidationHelper(s) createNumericConstraint(int, int, String, String) factory method.

```


SSFDataValidationConstraint dvConstraint = (XSSFDataValidationConstraint)
 dvHelper.createNumericConstraint(
   XSSFDataValidationConstraint.ValidationType.INTEGER,
   XSSFDataValidationConstraint.OperatorType.BETWEEN,
   "10", "100");
```

The values passed to the final two String parameters can be formulas; the '=' symbol is used to denote a formula. Thu the following would create a validation the allows values only if they fall between the results of summing two cell ran

```

XSSFDataValidationConstraint dvConstraint = (XSSFDataValidationConstraint)
  dvHelper.createNumericConstraint(
    XSSFDataValidationConstraint.ValidationType.INTEGER,
    XSSFDataValidationConstraint.OperatorType.BETWEEN,
    "=SUM(A1:A10)", "=SUM(B24:B27)");
```

It is not possible to create a drop down list if the createNumericConstraint() method is called, the setSuppressDropDownArrow(true) method call will simply be ignored.

Please check the javadoc for other constraint types as examples for those will not be included here. There are, for example, methods defined on the XSSFDataValidationHelper class allowing you to create the following types of constraint; date, time, decimal, integer, numeric, formula, text length and custom constraints.

Creating Data Validations From Spread Sheet Cells:

One other type of constraint not mentioned above is the formula list constraint. It allows you to create a validation th takes it value(s) from a range of cells. This code

```

XSSFDataValidationConstraint dvConstraint = (XSSFDataValidationConstraint)
    dvHelper.createFormulaListConstraint("$A$1:$F$1");
```

would create a validation that took it's values from cells in the range A1 to F1.

The usefulness of this technique can be extended if you use named ranges like this;

```

XSSFName name = workbook.createName();
name.setNameName("data");
name.setRefersToFormula("$B$1:$F$1");
XSSFDataValidationHelper dvHelper = new XSSFDataValidationHelper(sheet);
XSSFDataValidationConstraint dvConstraint = (XSSFDataValidationConstraint)
  dvHelper.createFormulaListConstraint("data");
CellRangeAddressList addressList = new CellRangeAddressList(
  0, 0, 0, 0);
```

```
XSSFDataValidation validation = (XSSFDataValidation)
  dvHelper.createValidation(dvConstraint, addressList);
validation.setSuppressDropDownArrow(true);
validation.setShowErrorBox(true);
sheet.addValidationData(validation);
```

OpenOffice Calc has slightly different rules with regard to the scope of names. Excel supports both Workbook and Sh
scope for a name but Calc does not, it seems only to support Sheet scope for a name. Thus it is often best to fully qual
the name for the region or area something like this;

```

XSSFName name = workbook.createName();
name.setNameName("data");
name.setRefersToFormula("'Data Validation'!$B$1:$F$1");
....
```

This does open a further, interesting opportunity however and that is to place all of the data for the validation(s) into named ranges of cells on a hidden sheet within the workbook. These ranges can then be explicitly identified in the setRefersToFormula() method argument.

ss.usermodel

The classes within the ss.usermodel package allow developers to create code that can be used to generate both binary (.xls) and SpreadsheetML (.xlsx) workbooks.

The techniques used to create data validations share much in common with the xssf.usermodel examples above. As a result just one or two examples will be presented here.

Check the value the user enters into a cell against one or more predefined value(s).

```

Workbook workbook = new XSSFWorkbook();  // or new HSSFWorkbook
Sheet sheet = workbook.createSheet("Data Validation");
DataValidationHelper dvHelper = sheet.getDataValidationHelper();
DataValidationConstraint dvConstraint =
dvHelper.createExplicitListConstraint(
  new String[]{"13", "23", "33"});
```

```
CellRangeAddressList addressList = new CellRangeAddressList(0, 0, 0, 0);
DataValidation validation = dvHelper.createValidation(
  dvConstraint, addressList);
// Note the check on the actual type of the DataValidation object.
// If it is an instance of the XSSFDataValidation class then the
// boolean value 'false' must be passed to the setSuppressDropDownArrow()
// method and an explicit call made to the setShowErrorBox() method.
if(validation instanceof XSSFDataValidation) {
  validation.setSuppressDropDownArrow(false);
  validation.setShowErrorBox(true);
}
else {
  // If the Datavalidation contains an instance of the HSSFDataValidation
  // class then 'true' should be passed to the setSuppressDropDownArrow()
  // method and the call to setShowErrorBox() is not necessary.
  validation.setSuppressDropDownArrow(true);
}
sheet.addValidationData(validation);
```

Drop Down Lists:

This code will do the same but offer the user a drop down list to select a value from.

```

Workbook workbook = new XSSFWorkbook();  // or new HSSFWorkbook
Sheet sheet = workbook.createSheet("Data Validation");
DataValidationHelper dvHelper = sheet.getDataValidationHelper();
DataValidationConstraint dvConstraint =
dvHelper.createExplicitListConstraint(
  new String[]{"13", "23", "33"});
CellRangeAddressList addressList = new CellRangeAddressList(0, 0, 0, 0);
DataValidation validation = dvHelper.createValidation(
  dvConstraint, addressList);
// Note the check on the actual type of the DataValidation object.
// If it is an instance of the XSSFDataValidation class then the
// boolean value 'false' must be passed to the setSuppressDropDownArrow()
// method and an explicit call made to the setShowErrorBox() method.
if(validation instanceof XSSFDataValidation) {
  validation.setSuppressDropDownArrow(true);
  validation.setShowErrorBox(true);
}
else {
  // If the Datavalidation contains an instance of the HSSFDataValidation
  // class then 'true' should be passed to the setSuppressDropDownArrow()
  // method and the call to setShowErrorBox() is not necessary.
  validation.setSuppressDropDownArrow(false);
}
sheet.addValidationData(validation);
```

Prompts and Error Messages:

These both exactly mirror the hssf.usermodel so please refer to the 'Messages On Error:' and 'Prompts:' sections abov

As the differences between the ss.usermodel and xssf.usermodel examples are small - restricted largely to the way the DataValidationHelper is obtained, the lack of any need to explicitly cast data types and the small difference in behaviour between the hssf and xssf interpretation of the setSuppressDropDowmArrow() method, no further exampl will be included in this section.

Advanced Data Validations.

Dependent Drop Down Lists.

In some cases, it may be necessary to present to the user a sheet which contains more than one drop down list. Furth the choice the user makes in one drop down list may affect the options that are presented to them in the second or subsequent drop down lists. One technique that may be used to implement this behaviour will now be explained.

There are two keys to the technique; one is to use named areas or regions of cells to hold the data for the drop down lists, the second is to use the INDIRECT() function to convert between the name and the actual addresses of the cells In the example section there is a complete working example- called LinkedDropDownLists.java - that demonstrates how to create linked or dependent drop down lists. Only the more relevant points are explained here.

To create two drop down lists where the options shown in the second depend upon the selection made in the first, be by creating a named region of cells to hold all of the data for populating the first drop down list. Next, create a data validation that will look to this named area for its data, something like this;

```

CellRangeAddressList addressList = new CellRangeAddressList(0, 0, 0, 0);
DataValidationHelper dvHelper = sheet.getDataValidationHelper();
DataValidationConstraint dvConstraint =
dvHelper.createFormulaListConstraint(
  "CHOICES");
DataValidation validation = dvHelper.createValidation(
  dvConstraint, addressList);
sheet.addValidationData(validation);
```

Note that the name of the area - in the example above it is 'CHOICES' - is simply passed to the createFormulaListConstraint() method. This is sufficient to cause Excel to populate the drop down list with data fron that named region.

Next, for each of the options the user could select in the first drop down list, create a matching named region of cells. The name of that region should match the text the user could select in the first drop down list. Note, in the example, a upper case letters are used in the names of the regions of cells.

Now, very similar code can be used to create a second, linked, drop down list;

```

CellRangeAddressList addressList = new CellRangeAddressList(0, 0, 1, 1);
DataValidationConstraint dvConstraint =
dvHelper.createFormulaListConstraint(
  "INDIRECT(UPPER($A$1))");
DataValidation validation = dvHelper.createValidation(
  dvConstraint, addressList);
```

```
sheet.addValidationData(validation);
```

The key here is in the following Excel function - INDIRECT(UPPER($A$1)) - which is used to populate the second, linked, drop down list. Working from the inner-most pair of brackets, it instructs Excel to look at the contents of cell to convert what it reads there into upper case – as upper case letters are used in the names of each region - and then convert this name into the addresses of those cells that contain the data to populate another drop down list.

Embedded Objects

It is possible to perform more detailed processing of an embedded Excel, Word or PowerPoint document, or to work with any other type of embedded object.

HSSF:

```

POIFSFileSystem fs = new POIFSFileSystem(new File("excel_with_embedded.xls"));
HSSFWorkbook workbook = new HSSFWorkbook(fs);
for (HSSFObjectData obj : workbook.getAllEmbeddedObjects()) {
    //the OLE2 Class Name of the object
    String oleName = obj.getOLE2ClassName();
    if (oleName.equals("Worksheet")) {
        DirectoryNode dn = (DirectoryNode) obj.getDirectory();
        HSSFWorkbook embeddedWorkbook = new HSSFWorkbook(dn, false);
        //System.out.println(entry.getName() + ": " + embeddedWorkbook.getNumberOfSheets());
    } else if (oleName.equals("Document")) {
        DirectoryNode dn = (DirectoryNode) obj.getDirectory();
        HWPFDocument embeddedWordDocument = new HWPFDocument(dn);
        //System.out.println(entry.getName() + ": " + embeddedWordDocument.getRange().text());
    }  else if (oleName.equals("Presentation")) {
        DirectoryNode dn = (DirectoryNode) obj.getDirectory();
        SlideShow<?,?> embeddedPowerPointDocument = new HSLFSlideShow(dn);
        //System.out.println(entry.getName() + ": " +
embeddedPowerPointDocument.getSlides().length);
    } else {
        if(obj.hasDirectoryEntry()){
            // The DirectoryEntry is a DocumentNode. Examine its entries to
find out what it is
            DirectoryNode dn = (DirectoryNode) obj.getDirectory();
            for (Entry entry : dn) {
                //System.out.println(oleName + "." + entry.getName());
            }
        } else {
            // There is no DirectoryEntry
            // Recover the object's data from the HSSFObjectData instance.
            byte[] objectData = obj.getObjectData();
        }
    }
}

```

XSSF:

```

XSSFWorkbook workbook = new XSSFWorkbook("excel_with_embeded.xlsx");
for (PackagePart pPart : workbook.getAllEmbeddedParts()) {
    String contentType = pPart.getContentType();
    // Excel Workbook - either binary or OpenXML
    if (contentType.equals("application/vnd.ms-excel")) {
        HSSFWorkbook embeddedWorkbook = new
HSSFWorkbook(pPart.getInputStream());
    }
    // Excel Workbook - OpenXML file format
    else if (contentType.equals("application/vnd.openxmlformats-officedocument.spreadsheetml.sheet")) {
        OPCPackage docPackage = OPCPackage.open(pPart.getInputStream());
        XSSFWorkbook embeddedWorkbook = new XSSFWorkbook(docPackage);
    }
    // Word Document - binary (OLE2CDF) file format
    else if (contentType.equals("application/msword")) {
        HWPFDocument document = new HWPFDocument(pPart.getInputStream());
    }
    // Word Document - OpenXML file format
    else if (contentType.equals("application/vnd.openxmlformats-officedocument.wordprocessingml.document")) {
        OPCPackage docPackage = OPCPackage.open(pPart.getInputStream());
        XWPFDocument document = new XWPFDocument(docPackage);
    }
    // PowerPoint Document - binary file format
    else if (contentType.equals("application/vnd.ms-powerpoint")) {
        HSLFSlideShow slideShow = new HSLFSlideShow(pPart.getInputStream());
    }
    // PowerPoint Document - OpenXML file format
    else if (contentType.equals("application/vnd.openxmlformats-officedocument.presentationml.presentation")) {
        OPCPackage docPackage = OPCPackage.open(pPart.getInputStream());
        XSLFSlideShow slideShow = new XSLFSlideShow(docPackage);
    }
    // Any other type of embedded object.
    else {
        System.out.println("Unknown Embedded Document: " + contentType);
        InputStream inputStream = pPart.getInputStream();
    }
```

```
}

```

(Since POI-3.7)

Autofilters

```

Workbook wb = new HSSFWorkbook(); //or new XSSFWorkbook();
Sheet sheet = wb.createSheet();
sheet.setAutoFilter(CellRangeAddress.valueOf("C5:F200"));

```

Conditional Formatting

```

Workbook workbook = new HSSFWorkbook(); // or new XSSFWorkbook();
Sheet sheet = workbook.createSheet();

SheetConditionalFormatting sheetCF = sheet.getSheetConditionalFormatting();

ConditionalFormattingRule rule1 =
sheetCF.createConditionalFormattingRule(ComparisonOperator.EQUAL, "0");
FontFormatting fontFmt = rule1.createFontFormatting();
fontFmt.setFontStyle(true, false);
fontFmt.setFontColorIndex(IndexedColors.DARK_RED.index);

BorderFormatting bordFmt = rule1.createBorderFormatting();
bordFmt.setBorderBottom(BorderStyle.THIN);
bordFmt.setBorderTop(BorderStyle.THICK);
bordFmt.setBorderLeft(BorderStyle.DASHED);
bordFmt.setBorderRight(BorderStyle.DOTTED);

```

```
PatternFormatting patternFmt = rule1.createPatternFormatting();
patternFmt.setFillBackgroundColor(IndexedColors.YELLOW.index);

ConditionalFormattingRule rule2 =
sheetCF.createConditionalFormattingRule(ComparisonOperator.BETWEEN, "-10",
"10");
ConditionalFormattingRule [] cfRules =
{
    rule1, rule2
};

CellRangeAddress[] regions = {
    CellRangeAddress.valueOf("A3:A5")
};

sheetCF.addConditionalFormatting(regions, cfRules);

```

See more examples on Excel conditional formatting in ConditionalFormats.java

Hiding and Un-Hiding Rows

Using Excel, it is possible to hide a row on a worksheet by selecting that row (or rows), right clicking once on the righ
hand mouse button and selecting 'Hide' from the pop-up menu that appears.

To emulate this using POI, simply call the setZeroHeight() method on an instance of either XSSFRow or HSSFRow (t
method is defined on the ss.usermodel.Row interface that both classes implement), like this:

```

Workbook workbook = new XSSFWorkbook();  // OR new HSSFWorkbook()
Sheet sheet = workbook.createSheet(0);
Row row = workbook.createRow(0);
row.setZeroHeight();

```

If the file were saved away to disc now, then the first row on the first sheet would not be visible.

Using Excel, it is possible to unhide previously hidden rows by selecting the row above and the row below the one tha is hidden and then pressing and holding down the Ctrl key, the Shift and the pressing the number 9 before releasing them all.

To emulate this behaviour using POI do something like this:

```

Workbook workbook = WorkbookFactory.create(new File(.......));
Sheet = workbook.getSheetAt(0);
Iterator<Row> row Iter = sheet.iterator();
while(rowIter.hasNext()) {
  Row row = rowIter.next();
  if(row.getZeroHeight()) {
    row.setZeroHeight(false);
  }
}

```

If the file were saved away to disc now, any previously hidden rows on the first sheet of the workbook would now be visible.

The example illustrates two features. Firstly, that it is possible to unhide a row simply by calling the setZeroHeight() method and passing the boolean value 'false'. Secondly, it illustrates how to test whether a row is hidden or not. Simp call the getZeroHeight() method and it will return 'true' if the row is hidden, 'false' otherwise.

Setting Cell Properties

Sometimes it is easier or more efficient to create a spreadsheet with basic styles and then apply special styles to certa cells such as drawing borders around a range of cells or setting fills for a region. CellUtil.setCellProperties lets you do that without creating a bunch of unnecessary intermediate styles in your spreadsheet.

Properties are created as a Map and applied to a cell in the following manner.

```

Workbook workbook = new XSSFWorkbook();  // OR new HSSFWorkbook()
```

```java
Sheet sheet = workbook.createSheet("Sheet1");
Map<String, Object> properties = new HashMap<String, Object>();

// border around a cell
properties.put(CellUtil.BORDER_TOP, BorderStyle.MEDIUM);
properties.put(CellUtil.BORDER_BOTTOM, BorderStyle.MEDIUM);
properties.put(CellUtil.BORDER_LEFT, BorderStyle.MEDIUM);
properties.put(CellUtil.BORDER_RIGHT, BorderStyle.MEDIUM);

// Give it a color (RED)
properties.put(CellUtil.TOP_BORDER_COLOR, IndexedColors.RED.getIndex());
properties.put(CellUtil.BOTTOM_BORDER_COLOR, IndexedColors.RED.getIndex());
properties.put(CellUtil.LEFT_BORDER_COLOR, IndexedColors.RED.getIndex());
properties.put(CellUtil.RIGHT_BORDER_COLOR, IndexedColors.RED.getIndex());

// Apply the borders to the cell at B2
Row row = sheet.createRow(1);
Cell cell = row.createCell(1);
CellUtil.setCellStyleProperties(cell, properties);

// Apply the borders to a 3x3 region starting at D4
for (int ix=3; ix <= 5; ix++) {
  row = sheet.createRow(ix);
  for (int iy = 3; iy <= 5; iy++) {
    cell = row.createCell(iy);
    CellUtil.setCellStyleProperties(cell, properties);
  }
}

```

NOTE: This does not replace the properties of the cell, it merges the properties you have put into the Map with the ce
existing style properties. If a property already exists, it is replaced with the new property. If a property does not exist,
is added. This method will not remove CellStyle properties.

Drawing Borders

In Excel, you can apply a set of borders on an entire workbook region at the press of a button. The PropertyTemplate
object simulates this with methods and constants defined to allow drawing top, bottom, left, right, horizontal, vertica
inside, outside, or all borders around a range of cells. Additional methods allow for applying colors to the borders.

It works like this: you create a PropertyTemplate object which is a container for the borders you wish to apply to a
sheet. Then you add borders and colors to the PropertyTemplate, and finally apply it to whichever sheets you need th
set of borders on. You can create multiple PropertyTemplate objects and apply them to a single sheet, or you can app
the same PropertyTemplate object to multiple sheets. It is just like a preprinted form.

Enums:

BorderStyle
: Defines the look of the border, is it thick or thin, solid or dashed, single or double. This enum replaces the CellStyle.BORDER_XXXXX constants which have been deprecated. The PropertyTemplate will not support th older style BORDER_XXXXX constants. A special value of BorderStyle.NONE will remove the border from a C once it is applied.

BorderExtent
: Describes the portion of the region that the BorderStyle will apply to. For example, TOP, BOTTOM, INSIDE, or OUTSIDE. A special value of BorderExtent.NONE will remove the border from the PropertyTemplate. When th template is applied, no change will be made to a cell border where no border properties exist in the PropertyTemplate.

```

// draw borders (three 3x3 grids)
PropertyTemplate pt = new PropertyTemplate();
// #1) these borders will all be medium in default color
pt.drawBorders(new CellRangeAddress(1, 3, 1, 3),
        BorderStyle.MEDIUM, BorderExtent.ALL);
// #2) these cells will have medium outside borders and thin inside borders
pt.drawBorders(new CellRangeAddress(5, 7, 1, 3),
        BorderStyle.MEDIUM, BorderExtent.OUTSIDE);
```

```
pt.drawBorders(new CellRangeAddress(5, 7, 1, 3), BorderStyle.THIN,
        BorderExtent.INSIDE);
// #3) these cells will all be medium weight with different colors for the
//     outside, inside horizontal, and inside vertical borders. The center
//     cell will have no borders.
pt.drawBorders(new CellRangeAddress(9, 11, 1, 3),
        BorderStyle.MEDIUM, IndexedColors.RED.getIndex(),
        BorderExtent.OUTSIDE);
pt.drawBorders(new CellRangeAddress(9, 11, 1, 3),
        BorderStyle.MEDIUM, IndexedColors.BLUE.getIndex(),
        BorderExtent.INSIDE_VERTICAL);
pt.drawBorders(new CellRangeAddress(9, 11, 1, 3),
        BorderStyle.MEDIUM, IndexedColors.GREEN.getIndex(),
        BorderExtent.INSIDE_HORIZONTAL);
pt.drawBorders(new CellRangeAddress(10, 10, 2, 2),
        BorderStyle.NONE,
        BorderExtent.ALL);

// apply borders to sheet
Workbook wb = new XSSFWorkbook();
Sheet sh = wb.createSheet("Sheet1");
pt.applyBorders(sh);

```

NOTE: The last pt.drawBorders() call removes the borders from the range by using BorderStyle.NONE. Like setCellStyleProperties, the applyBorders method merges the properties of a cell style, so existing borders are changed only if they are replaced by something else, or removed only if they are replaced by BorderStyle.NONE. To remove a color from a border, use IndexedColor.AUTOMATIC.getIndex().

Additionally, to remove a border or color from the PropertyTemplate object, use BorderExtent.NONE.

This does not work with diagonal borders yet.

Creating a Pivot Table

Pivot Tables are a powerful feature of spreadsheet files. You can create a pivot table with the following piece of code.

```

XSSFWorkbook wb = new XSSFWorkbook();
XSSFSheet sheet = wb.createSheet();

//Create some data to build the pivot table on
setCellData(sheet);

XSSFPivotTable pivotTable = sheet.createPivotTable(new
AreaReference("A1:D4"), new CellReference("H5"));
//Configure the pivot table
//Use first column as row label
pivotTable.addRowLabel(0);
//Sum up the second column
pivotTable.addColumnLabel(DataConsolidateFunction.SUM, 1);
//Set the third column as filter
pivotTable.addColumnLabel(DataConsolidateFunction.AVERAGE, 2);
//Add filter on forth column
pivotTable.addReportFilter(3);

```

Cells with multiple styles (Rich Text Strings)

To apply a single set of text formatting (colour, style, font etc) to a cell, you should create a CellStyle for the workbool
then apply to the cells.

```

// HSSF Example
HSSFCell hssfCell = row.createCell(idx);
//rich text consists of two runs
```

```
HSSFRichTextString richString = new HSSFRichTextString( "Hello, World!" );
richString.applyFont( 0, 6, font1 );
richString.applyFont( 6, 13, font2 );
hssfCell.setCellValue( richString );


// XSSF Example
XSSFCell cell = row.createCell(1);
XSSFRichTextString rt = new XSSFRichTextString("The quick brown fox");

XSSFFont font1 = wb.createFont();
font1.setBold(true);
font1.setColor(new XSSFColor(new java.awt.Color(255, 0, 0)));
rt.applyFont(0, 10, font1);

XSSFFont font2 = wb.createFont();
font2.setItalic(true);
font2.setUnderline(XSSFFont.U_DOUBLE);
font2.setColor(new XSSFColor(new java.awt.Color(0, 255, 0)));
rt.applyFont(10, 19, font2);

XSSFFont font3 = wb.createFont();
font3.setColor(new XSSFColor(new java.awt.Color(0, 0, 255)));
rt.append(" Jumped over the lazy dog", font3);

cell.setCellValue(rt);

```

To apply different formatting to different parts of a cell, you need to use RichTextString, which permits styling of par
of the text within the cell.

There are some slight differences between HSSF and XSSF, especially around font colours (the two formats store colours quite differently internally), refer to the HSSF Rich Text String and XSSF Rich Text String javadocs for more details.