Busy Developers' Guide to HSSF and XSSF Features

1. 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-depth coverage of the HSSF and XSSF user-APIs, please consult the HOWTO guide as it contains actual descriptions of how to use this stuff.

1.1. Index of Features

1.2. Features

1.2.1. New Workbook

```
Workbook wb = new HSSFWorkbook();
FileOutputStream fileOut = new FileOutputStream("workbook.xls");
wb.write(fileOut);
fileOut.close();

Workbook wb = new XSSFWorkbook();
FileOutputStream fileOut = new FileOutputStream("workbook.xlsx");
wb.write(fileOut);
fileOut.close();
```

1.2.2. New Sheet

```
Workbook wb = new HSSFWorkbook();
//Workbook wb = new XSSFWorkbook();
Sheet sheet1 = wb.createSheet("new sheet");
Sheet sheet2 = wb.createSheet("second sheet");
FileOutputStream fileOut = new FileOutputStream("workbook.xls");
wb.write(fileOut);
fileOut.close();
```

1.2.3. Creating Cells

```
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((short)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
FileOutputStream fileOut = new FileOutputStream("workbook.xls");
wb.write(fileOut);
fileOut.close();
```

1.2.4. Creating Date Cells

```
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
FileOutputStream fileOut = new FileOutputStream("workbook.xls");
wb.write(fileOut);
fileOut.close();
```

1.2.5. Working with different types of cells

```
Workbook wb = new HSSFWorkbook();
Sheet sheet = wb.createSheet("new sheet");
Row row = sheet.createRow((short)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(HSSFCell.CELL_TYPE_ERROR);

// Write the output to a file
FileOutputStream fileOut = new FileOutputStream("workbook.xls");
wb.write(fileOut);
fileOut.close();
```

1.2.6. 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((short) 2);
    row.setHeightInPoints(30);

    createCell(wb, row, (short) 0, XSSFCellStyle.ALIGN_CENTER, XSSFCellStyle.VERTIC
    createCell(wb, row, (short) 1, XSSFCellStyle.ALIGN_CENTER_SELECTION, XSSFCellSt
    createCell(wb, row, (short) 2, XSSFCellStyle.ALIGN_FILL, XSSFCellStyle.VERTICAL
    createCell(wb, row, (short) 3, XSSFCellStyle.ALIGN_GENERAL, XSSFCellStyle.VERTI
    createCell(wb, row, (short) 4, XSSFCellStyle.ALIGN_JUSTIFY, XSSFCellStyle.VERTI
    createCell(wb, row, (short) 5, XSSFCellStyle.ALIGN_LEFT, XSSFCellStyle.VERTICAL
    createCell(wb, row, (short) 6, XSSFCellStyle.ALIGN_RIGHT, XSSFCellStyle.VERTICA

    // Write the output to a file
    FileOutputStream fileOut = new FileOutputStream("xssf-align.xlsx");
    wb.write(fileOut);
    fileOut.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.
 */
```

```
    private static void createCell(Workbook wb, Row row, short column, short halign, sh
        Cell cell = row.createCell(column);
        cell.setCellValue(new XSSFRichTextString("Align It"));
        CellStyle cellStyle = wb.createCellStyle();
        cellStyle.setAlignment(halign);
        cellStyle.setVerticalAlignment(valign);
        cell.setCellStyle(cellStyle);
    }
```

1.2.7. 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(CellStyle.BORDER_THIN);
    style.setBottomBorderColor(IndexedColors.BLACK.getIndex());
    style.setBorderLeft(CellStyle.BORDER_THIN);
    style.setLeftBorderColor(IndexedColors.GREEN.getIndex());
    style.setBorderRight(CellStyle.BORDER_THIN);
    style.setRightBorderColor(IndexedColors.BLUE.getIndex());
    style.setBorderTop(CellStyle.BORDER_MEDIUM_DASHED);
    style.setTopBorderColor(IndexedColors.BLACK.getIndex());
    cell.setCellStyle(style);

    // Write the output to a file
    FileOutputStream fileOut = new FileOutputStream("workbook.xls");
    wb.write(fileOut);
    fileOut.close();
```

1.2.8. Iterate over rows and cells

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

Luckily, this is very easy. Row defines a *CellIterator* inner class to handle iterating over the cells (get one with a call to *row.cellIterator()*), and Sheet provides a *rowIterator()* method to give an iterator over all the rows.

Alternately, Sheet and Row both implement java.lang.Iterable, so using Java 1.5 you can simply take advantage of the built in "foreach" support - see below.

```
        Sheet sheet = wb.getSheetAt(0);
        for (Iterator<Row> rit = sheet.rowIterator(); rit.hasNext(); ) {
                Row row = rit.next();
                for (Iterator<Cell> cit = row.cellIterator(); cit.hasNext(); ) {
                        Cell cell = cit.next();
                        // Do something here
                }
        }
```

1.2.9. Iterate over rows and cells using Java 1.5 foreach loops

Sometimes, you'd like to just iterate over all the rows in a sheet, or all the cells in a row. If you are using Java 5 or later, then this is especially handy, as it'll allow the new foreach loop support to work.

Luckily, this is very easy. Both Sheet and Row implement *java.lang.Iterable* to allow foreach loops. For Row this allows access to the *CellIterator* inner class to handle iterating over the cells, and for Sheet gives the *rowIterator()* to iterator over all the rows.

```
        Sheet sheet = wb.getSheetAt(0);
        for (Row row : sheet) {
                for (Cell cell : row) {
                        // Do something here
                }
        }
```

1.2.10. 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.*;

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

                switch(cell.getCellType()) {
      case Cell.CELL_TYPE_STRING:
```

```
        System.out.println(cell.getRichStringCellValue().getString());
        break;
      case Cell.CELL_TYPE_NUMERIC:
        if(DateUtil.isCellDateFormatted(cell)) {
          System.out.println(cell.getDateCellValue());
        } else {
          System.out.println(cell.getNumericCellValue());
        }
        break;
      case Cell.CELL_TYPE_BOOLEAN:
        System.out.println(cell.getBooleanCellValue());
        break;
      case Cell.CELL_TYPE_FORMULA:
        System.out.println(cell.getCellFormula());
        break;
      default:
        System.out.println();
              }
        }
}
```

1.2.11. Text Extraction

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

```
        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();
```

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

1.2.12. 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((short) 1);

    // Aqua background
    CellStyle style = wb.createCellStyle();
    style.setFillBackgroundColor(IndexedColors.AQUA.getIndex());
    style.setFillPattern(CellStyle.BIG_SPOTS);
```

```
    Cell cell = row.createCell((short) 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(CellStyle.SOLID_FOREGROUND);
    cell = row.createCell((short) 2);
    cell.setCellValue("X");
    cell.setCellStyle(style);

    // Write the output to a file
    FileOutputStream fileOut = new FileOutputStream("workbook.xls");
    wb.write(fileOut);
    fileOut.close();
```

1.2.13. Merging cells

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

    Row row = sheet.createRow((short) 1);
    Cell cell = row.createCell((short) 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
    FileOutputStream fileOut = new FileOutputStream("workbook.xls");
    wb.write(fileOut);
    fileOut.close();
```

1.2.14. 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
    FileOutputStream fileOut = new FileOutputStream("workbook.xls");
    wb.write(fileOut);
    fileOut.close();
```

Note, the maximum number of unique fonts in a workbook is limited to 32767 (the maximum positive short). You should re-use fonts in your apllications 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((short) 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((short) 0);
            cell.setCellStyle(style);
        }
```

1.2.15. Custom colors

HSSF:

```
    HSSFWorkbook wb = new HSSFWorkbook();
    HSSFSheet sheet = wb.createSheet();
    HSSFRow row = sheet.createRow((short) 0);
    HSSFCell cell = row.createCell((short) 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(HSSFCellStyle.SOLID_FOREGROUND);

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

    cell.setCellStyle(style);

    //save with the default palette
    FileOutputStream out = new FileOutputStream("default_palette.xls");
    wb.write(out);
    out.close();

    //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
    out = new FileOutputStream("modified_palette.xls");
    wb.write(out);
    out.close();
```

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)));
    style1.setFillPattern(CellStyle.SOLID_FOREGROUND);
```

1.2.16. Reading and Rewriting Workbooks

```
    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(Cell.CELL_TYPE_STRING);
    cell.setCellValue("a test");

    // Write the output to a file
    FileOutputStream fileOut = new FileOutputStream("workbook.xls");
    wb.write(fileOut);
    fileOut.close();
```

1.2.17. 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 accomodate two lines of text
    row.setHeightInPoints((2*sheet.getDefaultRowHeightInPoints()));

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

    FileOutputStream fileOut = new FileOutputStream("ooxml-newlines.xlsx");
    wb.write(fileOut);
```

```
fileOut.close();
```

1.2.18. Data Formats

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

FileOutputStream fileOut = new FileOutputStream("workbook.xls");
wb.write(fileOut);
fileOut.close();
```

1.2.19. 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.

FileOutputStream fileOut = new FileOutputStream("workbook.xls");
wb.write(fileOut);
fileOut.close();
```

1.2.20. 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
);

FileOutputStream fileOut = new FileOutputStream("workbook.xls");
wb.write(fileOut);
fileOut.close();
```

1.2.21. Set Page Numbers on Footer

```
HSSFWorkbook wb = new HSSFWorkbook();
HSSFSheet sheet = wb.createSheet("format sheet");
HSSFFooter footer = sheet.getFooter()

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



// Create various cells and rows for spreadsheet.

FileOutputStream fileOut = new FileOutputStream("workbook.xls");
wb.write(fileOut);
fileOut.close();
```

1.2.22. Using the Convenience Functions

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

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

// Create a merged region
HSSFRow row = sheet1.createRow( (short) 1 );
HSSFRow row2 = sheet1.createRow( (short) 2 );
HSSFCell cell = row.createCell( (short) 1 );
```

```
cell.setCellValue( "This is a test of merging" );
Region region = new Region( 1, (short) 1, 4, (short) 4 );
sheet1.addMergedRegion( region );

// Set the border and border colors.
final short borderMediumDashed = HSSFCellStyle.BORDER_MEDIUM_DASHED;
HSSFRegionUtil.setBorderBottom( borderMediumDashed,
    region, sheet1, wb );
HSSFRegionUtil.setBorderTop( borderMediumDashed,
    region, sheet1, wb );
HSSFRegionUtil.setBorderLeft( borderMediumDashed,
    region, sheet1, wb );
HSSFRegionUtil.setBorderRight( borderMediumDashed,
    region, sheet1, wb );
HSSFRegionUtil.setBottomBorderColor(HSSFColor.AQUA.index, region, sheet1, wb);
HSSFRegionUtil.setTopBorderColor(HSSFColor.AQUA.index, region, sheet1, wb);
HSSFRegionUtil.setLeftBorderColor(HSSFColor.AQUA.index, region, sheet1, wb);
HSSFRegionUtil.setRightBorderColor(HSSFColor.AQUA.index, region, sheet1, wb);

// Shows some usages of HSSFCellUtil
HSSFCellStyle style = wb.createCellStyle();
style.setIndention((short)4);
HSSFCellUtil.createCell(row, 8, "This is the value of the cell", style);
HSSFCell cell2 = HSSFCellUtil.createCell( row2, 8, "This is the value of the cell")
HSSFCellUtil.setAlignment(cell2, wb, HSSFCellStyle.ALIGN_CENTER);

// Write out the workbook
FileOutputStream fileOut = new FileOutputStream( "workbook.xls" );
wb.write( fileOut );
fileOut.close();
```

1.2.23. 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);
```

1.2.24. Set a sheet as selected

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

1.2.25. 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(3,4);   // 75 percent magnification
```

1.2.26. 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 pains appear differently. The split area is divided into four separate work area's. The split occurs at the pixel level 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
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 );

FileOutputStream fileOut = new FileOutputStream("workbook.xls");
wb.write(fileOut);
fileOut.close();
```

1.2.27. Repeating rows and columns

It's possible to set up repeating rows and columns in your printouts by using the setRepeatingRowsAndColumns() function in the HSSFWorkbook class.

This function Contains 5 parameters. The first parameter is the index to the sheet (0 = first sheet). The second and third parameters specify the range for the columns to repeat. To stop the columns from repeating pass in -1 as the start and end column. The fourth and fifth parameters specify the range for the rows to repeat. To stop the columns from repeating pass in -1 as the start and end rows.

```
Workbook wb = new HSSFWorkbook();
Sheet sheet1 = wb.createSheet("new sheet");
Sheet sheet2 = wb.createSheet("second sheet");

// Set the columns to repeat from column 0 to 2 on the first sheet
wb.setRepeatingRowsAndColumns(0,0,2,-1,-1);
// Set the the repeating rows and columns on the second sheet.
wb.setRepeatingRowsAndColumns(1,4,5,1,2);

FileOutputStream fileOut = new FileOutputStream("workbook.xls");
wb.write(fileOut);
fileOut.close();
```

1.2.28. 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 f

FileOutputStream fileOut = new FileOutputStream("workbook.xls");
```

```
    wb.write(fileOut);
    fileOut.close();
```

1.2.29. Drawing Shapes

POI supports drawing shapes using the Microsoft Office drawing tools. Shapes on a sheet are organized in a hiearchy of groups and and shapes. The top-most shape is the patriarch. This is not visisble 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 converning 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.

Note:

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((short)3,3,500,500);
HSSFSimpleShape shape2 = group.createShape(new HSSFChildAnchor((short)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 it's own coordinate space for shapes that are placed 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 it's own coordinate space.

1.2.30. Styling Shapes

By default shapes can look a little plain. It's possible to apply different styles to the shapes however. The sorts of things 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);
```

1.2.31. 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`.

Note:

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(g
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 it's coordinates to match what we plan to draw. Next we calculate a reasonable fontSizeMultipler 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 it's 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 known 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).

1.2.32. Outlining

Outlines are great for grouping sections of information together and can be added easily to columns and rows using the 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( (short)4, (short)7 );
sheet1.groupColumn( (short)9, (short)12 );
sheet1.groupColumn( (short)10, (short)11 );

FileOutputStream fileOut = new FileOutputStream(filename);
wb.write(fileOut);
fileOut.close();
```

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

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

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

2. 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 a 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";
    FileOutputStream fileOut = new FileOutputStream(file);
    wb.write(fileOut);
    fileOut.close();
```

Note:

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

Reading images from a workbook:

```
    List lst = workbook.getAllPictures();
    for (Iterator it = lst.iterator(); it.hasNext(); ) {
        PictureData pict = (PictureData)it.next();
```

```
        String ext = pict.suggestFileExtension();
        byte[] data = pict.getData();
        if (ext.equals("jpeg")){
          FileOutputStream out = new FileOutputStream("pict.jpg");
          out.write(data);
          out.close();
        }
    }
```

3. 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 range. When working with Named Ranges, the classes: org.apache.poi.hssf.util.CellReference and & org.apache.poi.hssf.util.AreaReference are used (these work for both XSSF and HSSF, despite the package name).

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((short) 0).setCellValue(cvalue);

// 1. create named range for a single cell using areareference
Name namedCell = wb.createName();
namedCell.setNameName(cname);
String reference = sname+"!A1:A1"; // area reference
namedCell.setRefersToFormula(reference);

// 2. create named range for a single cell using cellreference
Name namedCel2 = wb.createName();
namedCel2.setNameName(cname);
String reference = sname+"!A1"; // cell reference
namedCel2.setRefersToFormula(reference);

// 3. create named range for an area using AreaReference
Name namedCel3 = wb.createName();
namedCel3.setNameName(cname);
String reference = sname+"!A1:C5"; // 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.getRefersToForm
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());
}
```

4. 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, but associated with, a cell

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

CreationHelper factory = wb.getCreationHelper();

Sheet sheet = wb.createSheet();

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

Drawing drawing = sheet.createDrawingPatriarch();

ClientAnchor anchor = factory.createClientAnchor();
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";
FileOutputStream out = new FileOutputStream(fname);
wb.write(out);
out.close();
```

Reading cell comments

```
Cell cell = sheet.get(3).getColumn((short)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);
```

5. Adjust column width to fit the contents

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

Note:

To calculate column width HSSFSheet.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` .

6. How to read hyperlinks

```
Sheet sheet = workbook.getSheetAt(0);

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

7. How to create hyperlinks

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

//cell style for hyperlinks
//by default hypelrinks 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((short)0);
cell.setCellValue("URL Link");

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

//link to a file in the current directory
```

```
    cell = sheet.createRow(1).createCell((short)0);
    cell.setCellValue("File Link");
    link = createHelper.createHyperlink(Hyperlink.LINK_FILE);
    link.setAddress("link1.xls");
    cell.setHyperlink(link);
    cell.setCellStyle(hlink_style);

    //e-mail link
    cell = sheet.createRow(2).createCell((short)0);
    cell.setCellValue("Email Link");
    link = createHelper.createHyperlink(Hyperlink.LINK_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((short)0).setCellValue("Target Cell");

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

    FileOutputStream out = new FileOutputStream("hyperinks.xlsx");
    wb.write(out);
    out.close();
```

8. Data Validations

Note:

Currently - as of version 3.5 - the XSSF stream does not support data validations and neither it nor the HSSF stream allow data validations to be recovered from sheets

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"});
```

```
HSSFDataValidation 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"});
HSSFDataValidation 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(HSSFDataValidation.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 as 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 for this method. The values passed to the two String parameters can be formulas; the '=' symbol is used to denote a formula

```
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

```
HSSFNamedRange 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 be given a name when created and that name should be used in the formula. So assuming the existence of a sheet named 'Data Sheet' this will work:

```
HSSFNamedRange 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:

```
HSSFNamedRange 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");
```

9. 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 FileInputStream("excel_with_embeded.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, fs, false);
        //System.out.println(entry.getName() + ": " + embeddedWorkbook.getNumberOfShe
    } else if (oleName.equals("Document")) {
        DirectoryNode dn = (DirectoryNode) obj.getDirectory();
        HWPFDocument embeddedWordDocument = new HWPFDocument(dn, fs);
        //System.out.println(entry.getName() + ": " + embeddedWordDocument.getRange()
    }  else if (oleName.equals("Presentation")) {
        DirectoryNode dn = (DirectoryNode) obj.getDirectory();
        SlideShow embeddedPowerPointDocument = new SlideShow(new HSLFSlideShow(dn, fs
        //System.out.println(entry.getName() + ": " + embeddedPowerPointDocument.getS
    } else {
        if(obj.hasDirectoryEntry()){
            // The DirectoryEntry is a DocumentNode. Examine its entries to find out
            DirectoryNode dn = (DirectoryNode) obj.getDirectory();
            for (Iterator entries = dn.getEntries(); entries.hasNext();) {
```

```
                Entry entry = (Entry) entries.next();
                //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.getAllEmbedds()) {
    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.spread
        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.wordpr
        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.presen
        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();
    }
}
```