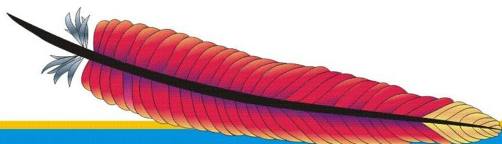
What's new with Apache POI

Apache POI - The Open Source Java solution for Microsoft Office

ApacheCon



Leading the Wave of Open Source

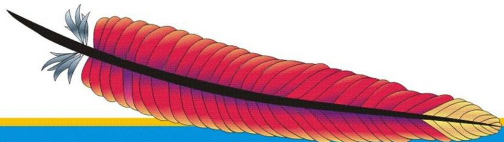
ApacheCon

Nick Burch

Senior Developer

Torchbox Ltd

TORCHBOX

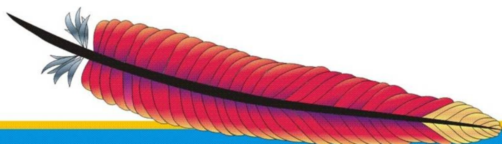


Leading the Wave of Open Source

What'll we be looking at?

- POI and OOXML – why bother?
- Working with Excel files
  - DOM-like UserModel for .xls and .xlsx
  - SAX-like EventModel for .xls
- Converting older code to the new style
- PowerPoint – ppt and pptx
- Word, Visio, Outlook, Publisher

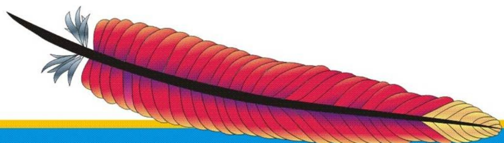
ApacheCon



Leading the Wave
of Open Source

But first... OLE2 & OOXML

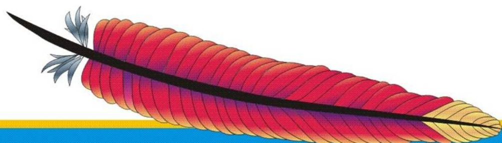
- All the old style file formats (xls, doc, ppt, vsd) are OLE2 based
- OLE2 is a binary format, that looks a lot like FAT
- All the new file formats (xlsx, docx, pptx) are OOXML
- OOXML is a zip file of XML files, which are data and metadata



OLE2 Overview

- OLE 2 Compound Document Format
- Binary file format, developed by Microsoft
- Contains different streams / files
- POIFS is our implementation of it
- http://poi.apache.org/poifs/fileformat.htr has the full details
- OOXML structure nods back to OLE2

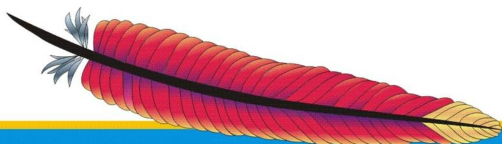
ApacheCon



Leading the Wave
of Open Source

Peeking inside OLE2

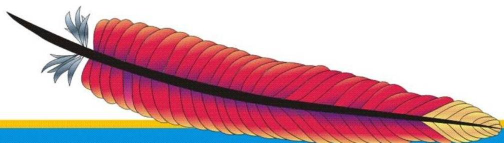
- POIFS has several tools for viewing and debugging OLE2 files
- org.apache.poi.poifs.dev.POIFSViewer is the main one
- Ant task is "POIFSViewer", pass it -Dfilename=src/testcase/....
- Documents have metadata entries, and streams for their data



ApacheCon
Leading the Wave of Open Source

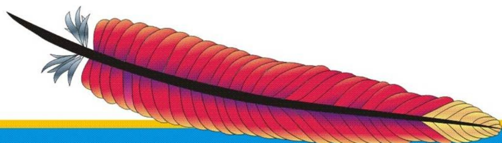
SimpleWithColour.xls

```
POIFS FileSystem
  Property: "Root Entry"
    Name              = "Root Entry"
  DocumentSummaryInformation
    Property: "DocumentSummaryInformation"
      Name              = "DocumentSummaryInformation"
      Document: "DocumentSummaryInformation" size=261
  SummaryInformation
    Property: "SummaryInformation"
      Name              = "SummaryInformation"
      Document: "SummaryInformation" size=229
  Workbook
    Property: "Workbook"
      Name              = "Workbook"
      Document: "Workbook" size=3177
```



OOXML Overview

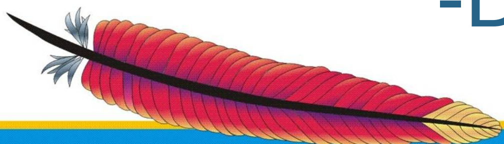
- Office Open XML file format
- Zip file of XML files
- Defined in ECMA standard 376
- Some files hold OOXML metadata, such as content type lookup
- Some files hold common office file format metadata, eg author
- Rest hold the document itself



ApacheCon
Leading the Wave of Open Source

Peeking inside OOXML

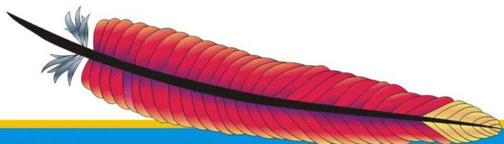
- You can just unzip the file to look at it
- The XML is saved without whitespace, so you'll want xmllint or similar to make it look nice
- POI also provides org.apache.poi.dev.OOXMLLister to make browsing easier
- Ant task is “OOXMLLister”, pass it
  -Dfilename=src/testcases/org/....



ApacheCon
Leading the Wave
of Open Source

SampleSS.xlsx

```
[Content_Types].xml

docProps/core.xml
docProps/app.xml

_rels/.rels

xl/_rels/workbook.xml.rels
xl/workbook.xml
xl/theme/theme1.xml
xl/worksheets/sheet2.xml
xl/worksheets/sheet3.xml
xl/worksheets/sheet1.xml
xl/sharedStrings.xml
xl/styles.xml
xl/calcChain.xml
```



SampleSS.xlsx cont.

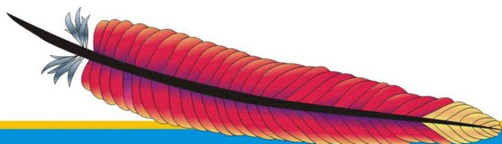
- src/testcases/org/apache/poi/hssf/data/SampleSS.xlsx
- [Content_Types].xml holds mapping between parts and their content types
- _rels/.rels holds relationships between parts of the file
- docProps/*.xml holds document metadata
- xl/*.xml holds excel data

ApacheCon
Leading the Wave
of Open Source

SampleSS .xls vs .xlsx

- .xls stores excel data in one big stream
- .xlsx stores excel data in 6+ streams, broken out by function
- Both have ways of storing document metadata
- .xlsx file is smaller, despite being xml based
- Processing .xlsx by hand is hard!

ApacheCon

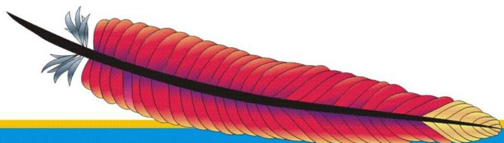


Leading the Wave
of Open Source

ApacheCon

Apache POI

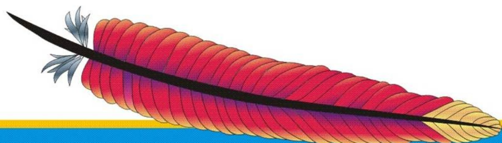
next section follows....



Leading the Wave
of Open Source

Excel – HSSF & XSSF

- Common UserModel (DOM-like) interface for both .xls and .xlsx
- Common UserModel evolved from existing HSSF UserModel
- Converting from HSSF to new style is fairly easy
- Getting started is even easier

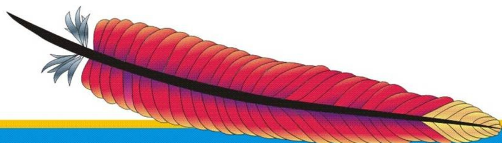


ApacheCon
Leading the Wave of Open Source

UserModel basics

- Open / create a Workbook
- From the Workbook, get existing or add new Sheets
- From a Sheet, get at the Rows
- From a Row, get at the Cells
- From a Cell, get contents (values), styling, formatting, comments etc

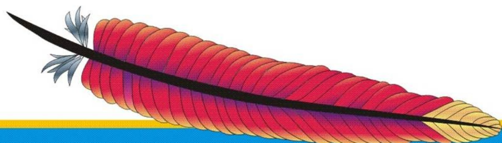
ApacheCon



Leading the Wave
of Open Source

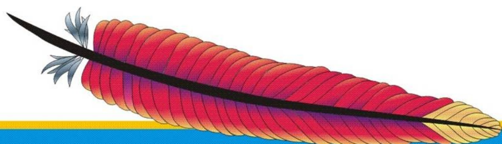
More on usermodel

- Object model fairly closely matched to the excel object model
- The odd bit of the low level file structure may peek through
- HSSF usermodel sits on top of model, which sits on top of records
- XSSF usermodel sits on top of XMLBeans, which does the xml



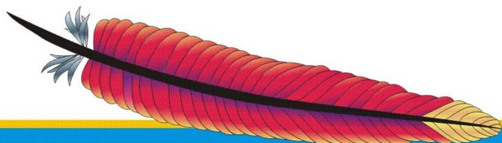
Cells are the key

- Cells contain data, and may optionally have stylings, formatting rules, comments, hyperlinks etc
- Cells can be of different types, such as string, number. Type can be important!
- Numbers and dates are always stored as floating point values
- Ints, dates etc are done by formatting these floating point values



org.apache.poi.ss.usermodel

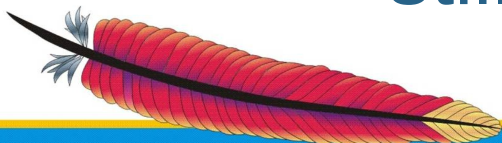
- Based on exsting HSSF UserModel, but now allows for both .xls and .xlsx support
- Fairly easy to covert existing code to it, to support both formats
- Unless you have very constrained memory, this is the thing to use



ApacheCon
Leading the Wave
of Open Source

org.apache.poi.ss cont.

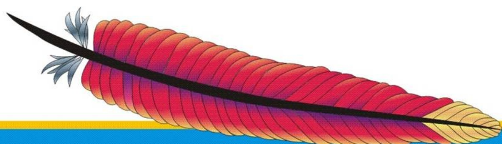
- Interfaces provide access to almost all the functionality you'll want for working with both types
- Where things are very different between OLE2 and OOXML, cast to the appropriate concrete class and use that
- Not everything supported yet for XSSF – still some stub methods



ApacheCon
Leading the Wave
of Open Source

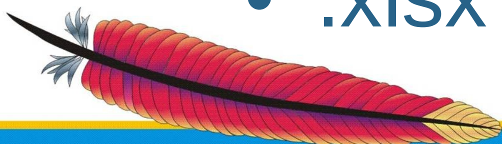
WorkbookFactory

- org.apache.poi.ss.usermodel.WorkbookFactory
- Handles creating the appropriate kind of [H|X]SSFWorkbook
- Accepts generic and implementation specific input sources
  Lets you code generically



Null Rows and Cells

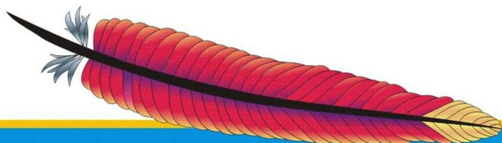
- If a row or cell has never had anything in it, Excel won't have saved it
- POI will give you a null if you ask for a row or a cell that doesn't exist in a file, so check your return values!
- Row and Cell iterators skip over any entries which don't exist, so there will be gaps
- .xlsx and.xls can differ slightly



A4 vs row=3, col=0

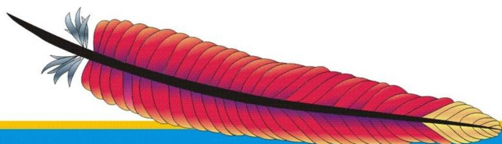
- Within the POI usermodel, we generally work with zero based row and cell numbers
- However, Excel uses 1 based row numbers, and A based column letters
- org.apache.poi.ss.util. CellReference will convert between the two for you

ApacheCon



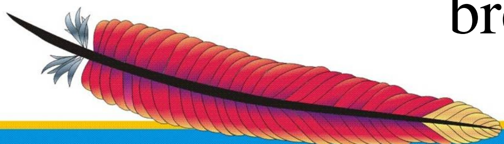
Dates in Excel

- Dates are stored as floating point numbers, which are fractional days since 1900 or 1904!
- No specific cell type in the file indicates that this is a date
- Only way to detect dates is from formatting rules
- POI tries to detect these for you



Fetching Cell Contents

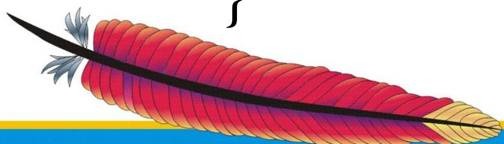
```
CellReference cellRef = new
        CellReference(r.getRowNum(), cell.getCellNum());
System.out.print(cellRef.formatAsString() + " - ");

// Need to switch on cell type, and call appropriate getter
switch(cell.getCellType()) {
    case Cell.CELL_TYPE_STRING:
        System.out.println(
            cell.getRichStringCellValue().getString());
        break;
    case Cell.CELL_TYPE_BOOLEAN:
        System.out.println(cell.getBooleanCellValue());
        break;
```



ApacheCon
Leading the Wave
of Open Source

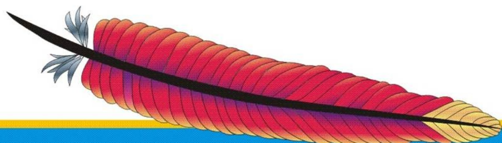
Fetching Cell Contents cont

```
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_FORMULA:
        System.out.println(cell.getCellFormula());
        break;
default:
        System.out.println();
}
```



A note on shorts

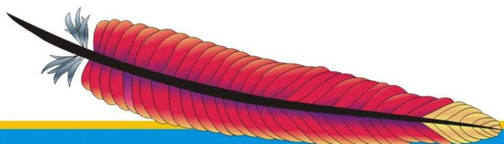
- Historically, POI had used shorts to address row and cell numbers
- Excel row and cell numbers map nicely into shorts
- But it can be confusing for users, especially on rows > 32768
- POI has largely moved to using ints, but the odd short remains



Looping over cells

- Loop over all sheets, using Workbook.getNumberofSheets() and Workbook.getSheetAt(idx)
- On Sheet, call rowIterator / use Java 1.5 foreach loop
- On Row, call cellIterator / use Java 1.5 foreach loop
- You now get all the Cells

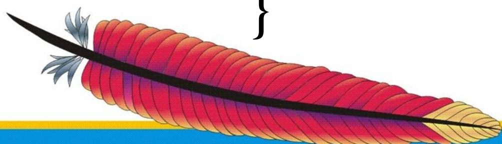
ApacheCon



Leading the Wave
of Open Source

Cell Looping Example

```
// If you want to test, use src/testcases/org/
//      apache/poi/hssf/data/SimpleWithColours.xls
Workbook wb = WorkbookFactory.create(
                    new FileInputStream("myfile.xls"));
for(int sn=0; sn<wb.getNumberOfSheets(); sn++) {
   Sheet sheet = wb.getSheetAt(sn);
   for(Row row : sheet) {
      for(Cell cell : row) {
         // Do something with the cell here
         System.out.println(cell.getCellType());
      }
   }
}
```

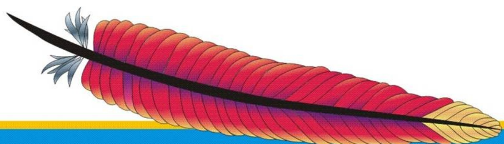


ApacheCon
Leading the Wave
of Open Source

Converting HSSF to SS

- Generally, hssf.usermodel.HSSFFoo has become ss.usermodel.Foo
- hssf.util.HSSFFoo has generally become ss.util.Foo
- Switch your code to use ss interfaces, rather than concrete classes
- When you would have new'd a HSSF, instead ask the CreateHelper to give you a new one

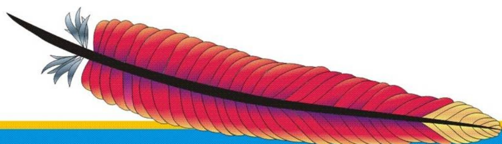
ApacheCon



Leading the Wave
of Open Source

Converting HSSF to SS

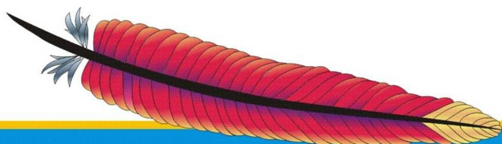
- Use WorkbookFactory when opening
- Use new H/X SSFWorkbook when creating a file from scratch
- If you want to do something file format specific, cast the generic interface to the appropriate HSSF / XSSF class



ApacheCon
Leading the Wave
of Open Source

Auto-Sizing Columns

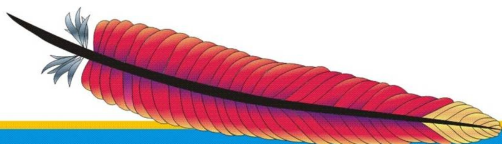
- Handled by the sheet
- Needs to use awt stuff, so either must be running in a graphical environment, or be correctly set to headless
- If running headless, first call System.setProperty(
      "java.awt.headless", "true")).
- Heavily dependent on correct fonts



ApacheCon
Leading the Wave
of Open Source

Cell Comments

- Cell comments are the little red triangles in the corners of cells, which can be expanded to show formatted text, commenting on the contents of the cell
- Support for both HSSF and XSSF
- Using Comments the same for both, but creating them differs



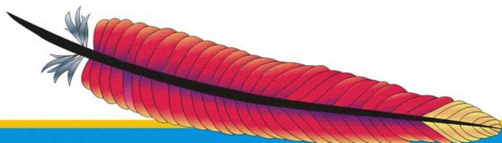
ApacheCon
Leading the Wave
of Open Source

HSSF Cell Comments

- Cell comments appear to have been kludged into OLE2 xls
- They are very similar to other drawing types, eg text boxes
- You create a comment from the base drawing Patriarch of your sheet (create if needed)
- Once created, set fields, and then attach to a cell

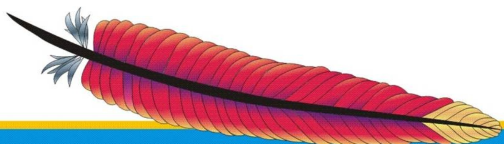
XSSF Cell Comments

- Much more nicely done for OOXML .xlsx than for OLE2 .xls
- Each sheet has another part, which holds all the comments for its cells
- You create a comment from a sheet
- Once created, set fields, and then attach to a cell



HSSF Comment example

```
// Assumes you have a sheet and a cell
HSSFPatriarch patr = sheet.createDrawingPatriarch();
// Anchor defines size and position of the comment
Comment comment1 = patr.createComment(
new HSSFClientAnchor(0, 0, 0, 0, (short)4, 2, (short) 6, 5));
// Set text and author of the comment
comment1.setString(new HSSFRichTextString(
        "We can set comments in POI"));
comment1.setAuthor(“Apache POI”);
// Assign to a particular cell
cell.setCellComment(comment1);
```

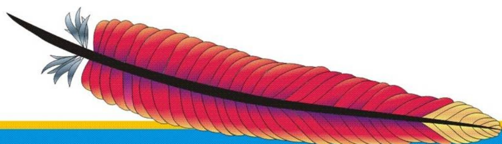


ApacheCon

Leading the Wave
of Open Source

XSSF Comment example

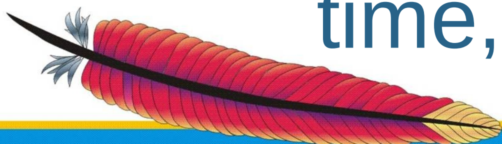
```
Sheet sheet = wb.getSheetAt(1);
Row row = sheet.getRow(2);
Cell cell = row.createCell(0);
// Create a new comment for this sheet
Comment comment1 = sheet.createComment();
// Set text and author of the comment
comment1.setString(new XSSFRichTextString(
        "We can set comments in POI for XSSF too"));
comment1.setAuthor(“Apache POI”);
// Assign to a particular cell
cell.setCellComment(comment1);
```



ApacheCon
Leading the Wave
of Open Source

Headers and Footers

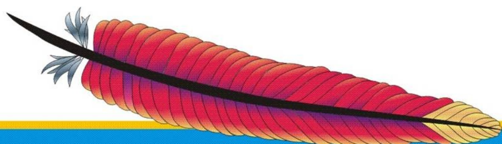
- HSSF and XSSF, but XSSF does more
- Works very similar for both Headers and Footers
- Controlled on a per-sheet basis
- Can set text for left edge, centre and right edge
- Special strings for current date, current time, page number etc



Headers and Footers cont

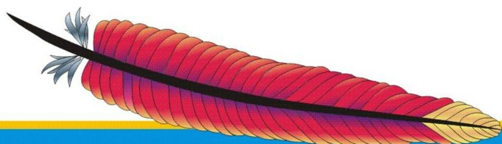
- Get with sheet.getHeader() or sheet.getFooter()
- Total text size is limited to 255 bytes long, across all areas
- Call methods on Header or Footer to get the magic strings for sheet number etc
- Only very limited text formatting

ApacheCon



Charts and Pictures

- With HSSF, it is not possible to edit charts or pictures
- (Ditto XSSF, but that ought to be easier to support)
- With HSSF, you can add new pictures, and some simple drawings, but you can't add charts



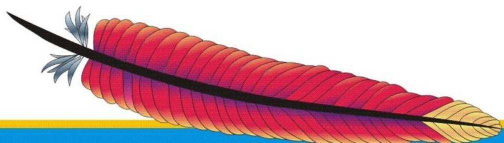
Charts Workaround

- We want a chart based on data POI will add in, but we can't create the chart with POI
- Instead, open empty file in excel, create chart, and point it at a simple Named Range
- With POI, put in all your data, then update the named range
- Excel will now chart new data

ApacheCon

Apache POI

next section follows....

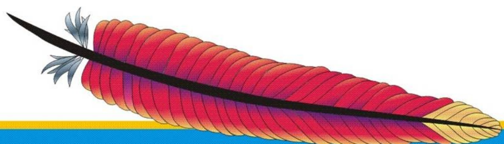


Leading the Wave
of Open Source

Excel Formulas

- Excel formulas are a bit tricky
- Cell holds both the formula, and a calculated value
- For .xls, formula is stored in a semi-parsed reverse polish format, with any data they need tacked on the end
- For .xlsx, the formula is just text

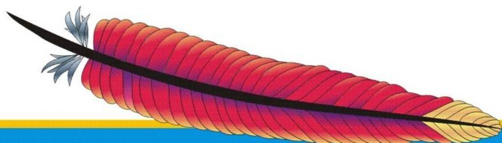
ApacheCon



Leading the Wave
of Open Source

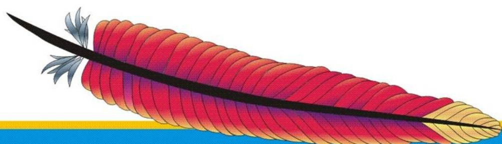
Formula Gotcha

- Open a file containing formulas
- Alter the value of a cell that a formula depends on
- Save the file
- Open in Excel
- Formula cell still has old value
- Open formula cell, hit enter
- Formula cell now gets new value



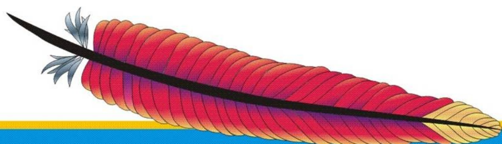
Formula Result Caching

- To make the spreadsheet load faster, Excel stores a computed value with every formula
- Excel will automatically re-calculate formulas for you when source data changes
- POI can re-calculate formulas, but you need to explicitly trigger this



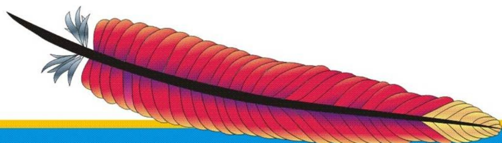
Formula Evaluator

- POI has a largely complete implementation of the Excel formula parser and evaluator
- Doesn't do some functions, but now does most
- Massive improvements of late
- Used to be in scratchpad, but recently moved into main codebase



Using the evaluator

- Evaluator class is org.apache.poi.ss.usermodel.FormulaEvaluator
- Has a variety of methods, but most common are:
  - evaluateFormulaCell(cell)
  - evaluateAllFormulaCells(workbook)
- Only run at very end of processing, just before saving

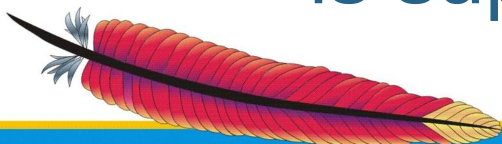


ApacheCon
Leading the Wave of Open Source

What is supported?

- Almost all basic functions are
- VBA based ones never will be
- Quite a few complex things are
- Some of the POI function classes are stubs – if it extends NotImplementedFunction it isn't done yet
- If not, but the class exists, it probably is supported

ApacheCon

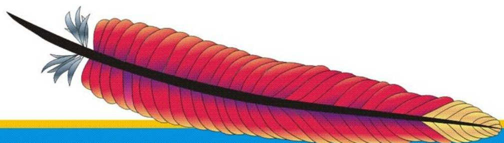


Leading the Wave
of Open Source

ApacheCon

Apache POI

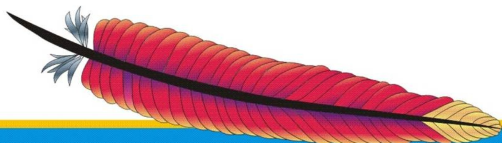
next section follows....



Leading the Wave
of Open Source

org.apache.poi.hssf.eventusermodel

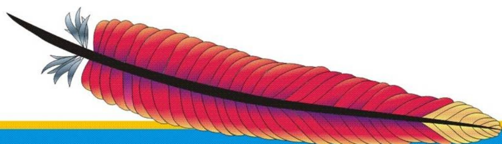
- Event based reading (no write)
- Basic form fires off for each record that is encountered
- Can filter to only give certain records
- Provides pull method too
- Only fires for records that exist in the file, but there is a workaround



ApacheCon
Leading the Wave of Open Source

All record based

- Only works with Records though, which are quite low level
- You may find yourself needing to refer to the Microsoft docs
- If in doubt, check the equivalent POI UserModel sourcecode, to see how that works with the records
- That said, it isn't too scary



ApacheCon
Leading the Wave
of Open Source

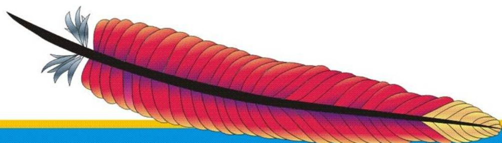
Key Records

- NumberRecord – number cell
- FormulaRecord – formula cell, with result if numeric
- LabelSSTRecord – string cell, points to a SSTRecord
- SSTRecord – holds shared strings
- StringRecord – string, used in other parts of the file
- FormatRecord – number format

ApacheCon
Leading the Wave of Open Source

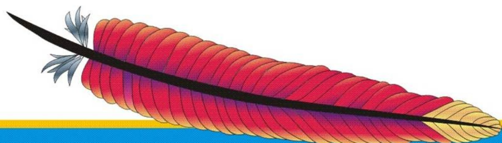
Record ordering

- Many key records come early in the file
- Sheet names and ordering are one such thing, as are all the strings used
- Each sheet's data starts with a BOFRecord
- All rows come before all cells



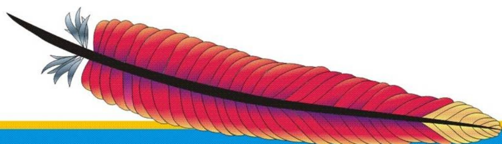
Using the EventModel

- Register a listener, and tell it what kinds of records you want (possibly all of them)
- HSSFEventFactory processes the file, calling your listener for all appropriate records
- You do something as the records come in



EventModel – in code

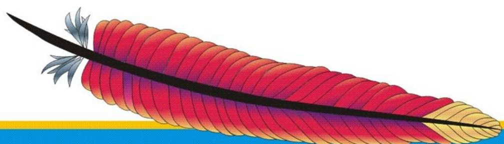
```
FileInputStream fin = new FileInputStream(filename);
POIFSFileSystem poifs = new POIFSFileSystem(fin);
// get the Workbook (excel part) stream
InputStream din =
poifs.createDocumentInputStream("Workbook");
// Request all records
HSSFRequest req = new HSSFRequest();
req.addListenerForAllRecords(new EventExample());
HSSFEventFactory factory = new HSSFEventFactory();
factory.processEvents(req, din);
din.close(); fin.close();
```



ApacheCon
Leading the Wave
of Open Source

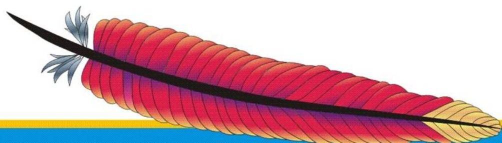
And a sample Listener

```
public class EventExample implements HSSFListener {
    public void processRecord(Record record) {
        switch (record.getSid()) {
            case BoundSheetRecord.sid:
                BoundSheetRecord bsr = (BoundSheetRecord) 
record;
                System.out.println("New sheet named: " + 
bsr.getSheetname());
                break;
        }
}}
```



“Missing” Records

- If you've never put data in a cell, or in a row, there won't be records for it
- But, POI can generate dummy records for you when there is a gap
- You get dummy records for missing row, missing cell, and last cell of a given row



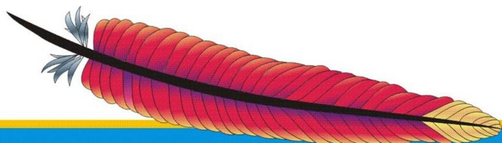
ApacheCon
Leading the Wave of Open Source

Getting Missing Records

- Create a org.apache.poi.hssf.eventusermodel.MissingRecordAwareHSSFListener
- Pass your own HSSFListener to this
- Have the normal eventmodel code run against the missing record listener
- MissingRecordAwareHSSFListener sits between your HSSFListener and the event factory / HSSFRequest

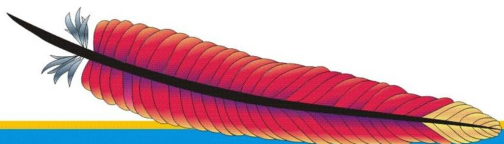
Missing Records Example

- Much too much code to fit onto a slide or two
- Handily located in svn
- /src/scratchpad/examples/src/org/apache/poi/hssf/eventusermodel/examples/XLS2CSVmra.java
- Converts a XLS file to CSV, without issues over gaps in the records



EventModel tips

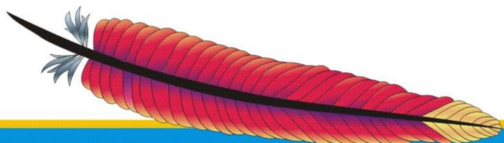
- Grab useful records as they come past, as they only come once
- But ignore all others, to keep processing and memory light
- If in doubt, check how UserModel does it
- If possible, just buy more memory and use UserModel!



ApacheCon
Leading the Wave of Open Source

Apache POI

next section follows....

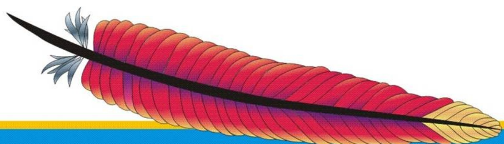


ApacheCon

Leading the Wave
of Open Source

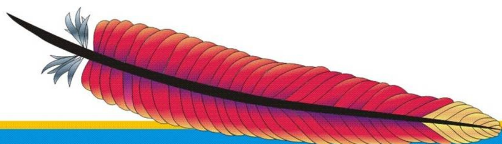
.xlsx and SAX

- Need to use POI to get at the bits of the file we're interested in
- Shared Strings table and Styles table makes life tricky, but POI can help
- Process each sheet individually
- Tends to only have non-blank entries, so there are gaps



Starting with SAX

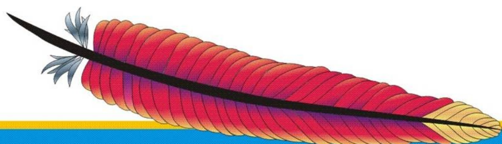
- Core POI class is org.apache.poi.xssf.eventusermodel.XSSFReader
- This provides a way to get at the raw data for the workbook, and all the sheets
- Also lets you get at shared strings table and styles table



ApacheCon
Leading the Wave
of Open Source

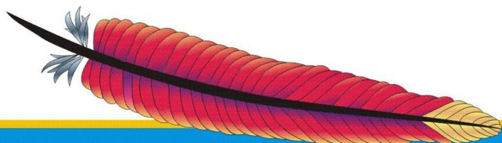
SAX + Shared Strings

- Most string cells actually just contain a pointer to the shared strings table, and not the string value itself
- You need to process the shared strings table to get the string contents
- Can just get the raw contents
- Or can get a handy helper



SAX + Styles

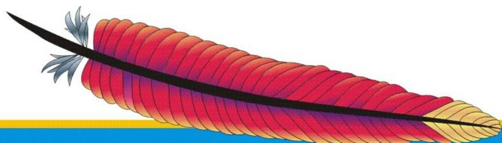
- Cells don't contain their styles, just a pointer to the styles table
- You need to process the styles table to get fonts, colours, number formatting rules etc
- Can just get the raw contents
- Or can get a handy helper



ApacheCon
Leading the Wave of Open Source

Handy Helpers

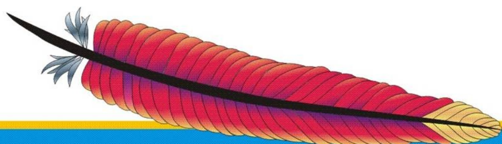
- XSSFReader.getSharedStringsTable() - lets you look up shared strings from their index
- XSSFReader.getStylesTable() - lets you look up styles, fonts, number formats etc. Go style to font, style to formatting rules etc



SAX + Sheets

- Two options, either get all sheets, or get a specific sheet
- To get a specific sheet, get the r:id from the workbook's definition of the sheet
- To get all sheets, there is an iterator
- Close InputStreams when done, to keep the memory footprint down

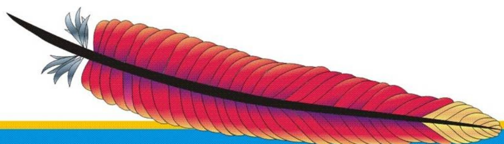
ApacheCon



Leading the Wave
of Open Source

SAX in action

```
XMLReader parser = fetchSheetParser();

Package pkg = Package.open(“myfile.xlsx”);
XSSFReader r = new XSSFReader( pkg );
SharedStringsTable sst = r.getSharedStringsTable();
Iterator<InputStream> sheets = r.getSheetsData();
while(sheets.hasNext()) {
	InputStream sheet = sheets.next();
	InputSource sheetSource = new InputSource(sheet);
	parser.parse(sheetSource);
	sheet.close();	}
```

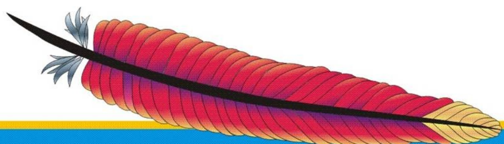


ApacheCon

Leading the Wave
of Open Source

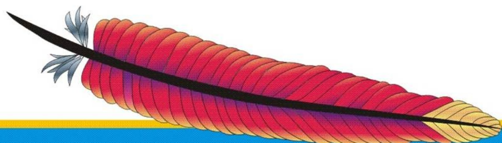
Getting an XMLReader

```
// Readers may be re-used, but only sequentually
public XMLReader fetchSheetParser() throws
                                        SAXException {
    XMLReader parser =
        XMLReaderFactory.createXMLReader(
            "org.apache.xerces.parsers.SAXParser"
    );
   ContentHandler handler = new SheetHandler();
   parser.setContentHandler(handler);
   return parser;
}
```



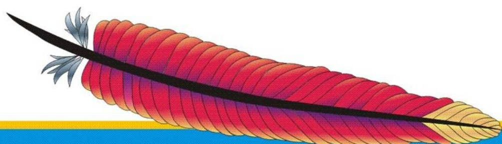
Simple ContentHandler

```
// See org.xml.sax.helpers.DefaultHandler javadocs
public class SheetHandler extends DefaultHandler {
    public void startElement(String uri, String localName,
                    String name, Attributes attributes)
                    throws SAXException {
        super.startElement(uri, localName, name, attributes);
        if(name.equals("c")) {
            System.out.println(attributes.getValue("r"));
        }
    }
}
```



SST Aware Handler

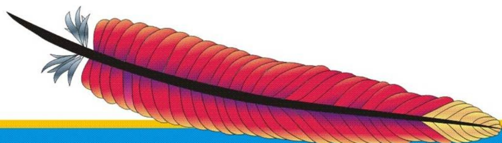
```
public  class SheetHandler extends DefaultHandler {
  private SheetHandler(SharedStringsTable sst) { this.sst = sst; }
  public void startElement(String u, String ln, String name, Attributes attr) throws Exception {
        if(name.equals("c")) { // c => cell
              // Print the cell reference
              System.out.print(attr.getValue("r") + " - ");
              // Figure out if the value is an index in the SST
              String cellType = attr.getValue("t");
              if(cellType != null && cellType.equals("s")) {
                    nextIsString = true;
              } else {
                    nextIsString = false;
              }
        } }
public void endElement(String u, String ln, String name)  throws SAXException {
        if(name.equals("v")) { // v => contents of a cell
              System.out.println(lastContents);
        } }
```



ApacheCon
Leading the Wave
of Open Source

SST Aware Handler cont

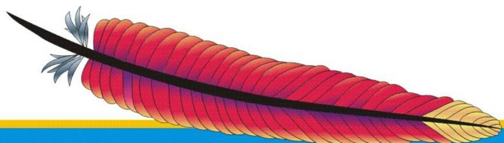
```
public void characters(char[] ch, int start, int length) throws SAXException {
        lastContents = new String(ch, start, length);
        if(nextIsString) {
                int idx = Integer.parseInt(lastContents);
                lastContents = sst.getSharedStringAt(idx);
        }
}

private SharedStringsTable sst;
private String lastContents;
private boolean nextIsString;
}
```



ApacheCon

Apache POI

next section follows....

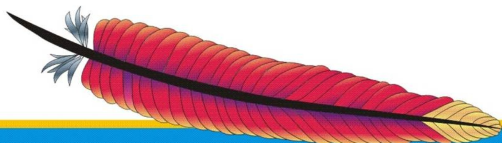


Leading the Wave
of Open Source

Text Extraction

- POI provides a large number of text extractors
- Most Office file formats are covered, to varying levels of completeness
- Metadata and embeded documents are supported too
- Tika might wrap everything you want up already though

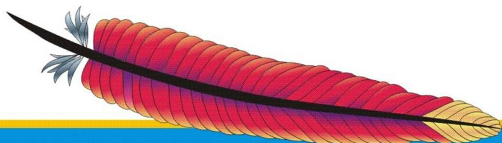
ApacheCon



Leading the Wave
of Open Source

Key Classes

- org.apache.poi.POITextExtractor
  base of all extractors. Lets you get text and metadata
- org.apache.poi.POIOLE2TextExtractor
  for older file formats
- org.apache.poi.POIXMLTextExtractor
  for newer, ooxml file formats

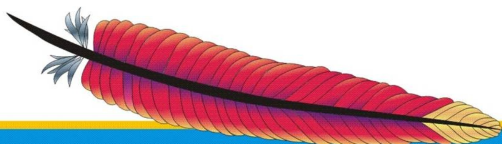


ApacheCon
Leading the Wave
of Open Source

Key Classes cont.

- org.apache.poi.extractor.
  ExtractorFactory

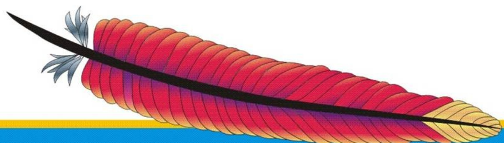
- Identifies the file format, and builds the appropriate extractor for you
  Able to identify embeded documents within your file, and give you extractors for those too



ApacheCon

Apache POI

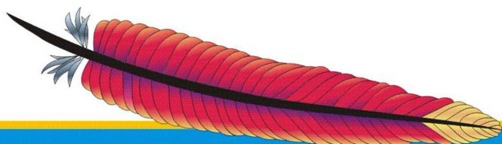
next section follows....



PowerPoint – HSLF & XSLF

- Existing code in POI Scratchpad partly converted to do .pptx too
- Code much less advanced than equivalents in HSSF/XSSF
- Mix of low-level and high-level code
- Much of the file format deals with Escher shapes and objects

ApacheCon

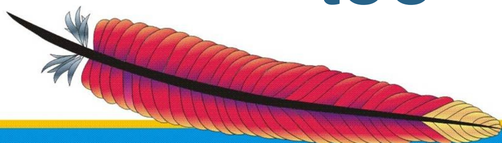


Leading the Wave
of Open Source

PowerPoint Text

- Slideshow is made up of a number of slides, which can optionally have a notes sheet associated with them
- Notes are fairly simple, and normally just have rich text
- Slides can have text runs directly, but normally have lots of Escher drawings, which may occasionally contain text too

ApacheCon



PowerPoint text continued

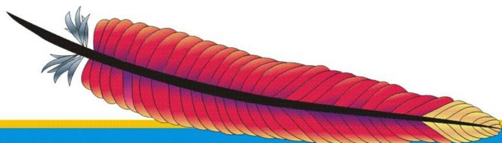
- From both a Slide and Notes, you can get TextRuns, which contain the text of the sheet, no matter where it lives
- On these, you can get the text, get the rich text formatting, change the text of change the formatting
- Adding new text is only supported via adding a new Escher text box onto the page, and adding text to that

ApacheCon

Leading the Wave
of Open Source

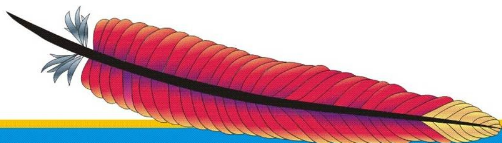
Supported Features

- Reading and changing text
- Adding new slides
- Drawing simple shapes onto slides
- Adding, changing and extracting pictures
- Hyperlinks
- Tables
- Viewing and editing slide backgrounds



PowerPoint Rendering

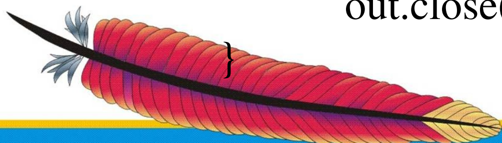
- Yegor has done some excellent work on rendering PowerPoint files into images
- HSLF only for now, but works well for simpler slides already
- http://poi.apache.org/slideshow/how-to-shapes.html#Graphics2D



ApacheCon
Leading the Wave of Open Source

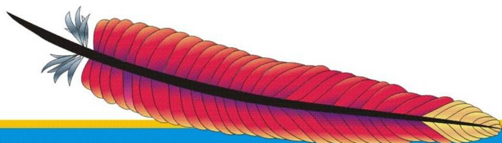
Rendering Example

```
FileInputStream is = new FileInputStream("slideshow.ppt");
SlideShow ppt = new SlideShow(is);

 Dimension pgsize = ppt.getPageSize();
 Slide[] slide = ppt.getSlides();
for (int i = 0; i < slide.length; i++) {
          BufferedImage img = new BufferedImage(pgsize.width, pgsize.height,
BufferedImage.TYPE_INT_RGB);
          Graphics2D graphics = img.createGraphics();
          //clear the drawing area
          graphics.setPaint(Color.white);
          graphics.fill(new Rectangle2D.Float(0, 0, pgsize.width, pgsize.height));
          //render
          slide[i].draw(graphics);
         //save the output
          FileOutputStream out = new FileOutputStream("slide-"  + (i+1) + ".png");
          javax.imageio.ImageIO.write(img, "png", out);
          out.close();
}
```



Creating PPT in Java

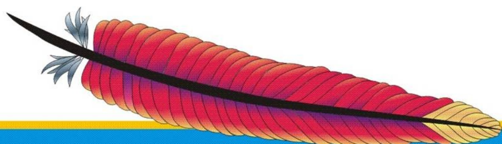
- HSLF only for now
- It's possible to treat a HSLF file almost as a Graphics2D canvas, and draw onto it
- See http://people.apache.org/~yegor/apachecon_eu08/ for a demo



ApacheCon
Leading the Wave
of Open Source

Drawing text

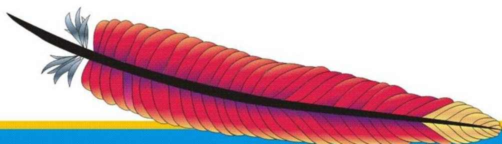
```
Slide slide = ppt.createSlide();

TextBox box1 = new TextBox();
TextRun tr1 = box1.getTextRun();
tr1.setRunType(TextHeaderAtom.TITLE_TYPE);
tr1.setText("Isn't HSLF excellent?");
box1.setAnchor(new Rectangle(36, 21, 648, 90));
slide.addShape(box1);

TextBox box2 = new TextBox();
TextRun tr2 = box2.getTextRun();
tr2.setRunType(TextHeaderAtom.BODY_TYPE);
tr2.setText("HorribleSLideshowFormat is the POI Project's pure Java implementation " +
        "of the Powerpoint '97(-2007) file format. \r" +
        "It supports drawing text new on a PPT page");
box2.setAnchor(new Rectangle(36, 126, 648, 356));
slide.addShape(box2);
```



ApacheCon
Leading the Wave
of Open Source

Drawing a bar graph

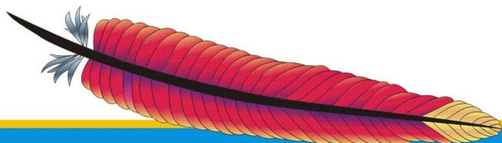
```
//bar chart data. The first value is the bar color, the second is the width
Object[] def = new Object[]{
    Color.yellow, new Integer(100),
    Color.green, new Integer(150),
    Color.gray, new Integer(75),
    Color.red, new Integer(200),
};

Slide slide = ppt.createSlide();

ShapeGroup group = new ShapeGroup();
//define position of the drawing in the slide
Rectangle bounds = new java.awt.Rectangle(200, 100, 350, 300);
group.setAnchor(bounds);
slide.addShape(group);
Graphics2D graphics = new PPGraphics2D(group);
```



ApacheCon
Leading the Wave
of Open Source

Drawing a bar graph cont.

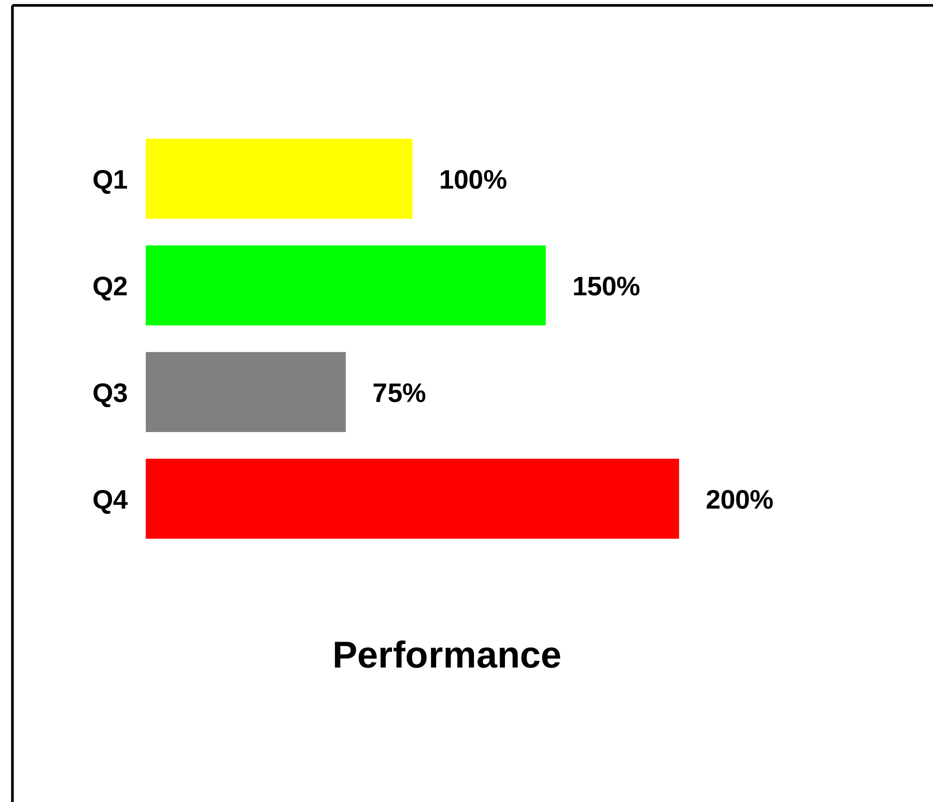
```
//draw a simple bar graph
int x = bounds.x + 50, y = bounds.y + 50;
graphics.setFont(new Font("Arial", Font.BOLD, 10));
for (int i = 0, idx = 1; i < def.length; i+=2, idx++) {
    graphics.setColor(Color.black);
    int width = ((Integer)def[i+1]).intValue();
    graphics.drawString("Q" + idx, x-20, y+20);
    graphics.drawString(width + "%", x + width + 10, y + 20);
    graphics.setColor((Color)def[i]);
    graphics.fill(new Rectangle(x, y, width, 30));
    y += 40;
}
graphics.setColor(Color.black);
graphics.setFont(new Font("Arial", Font.BOLD, 14));
graphics.draw(bounds);
graphics.drawString("Performance", x + 70, y + 40);
```

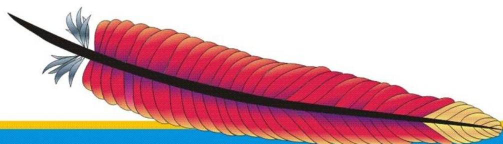


ApacheCon

Leading the Wave
of Open Source

Result – we can draw!

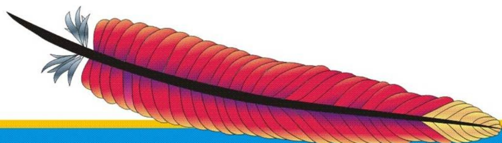




ApacheCon

Apache POI

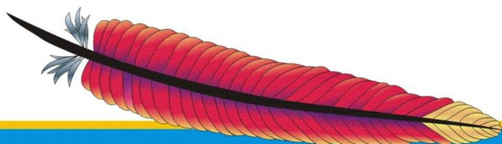
next section follows....



Leading the Wave
of Open Source

Word – HWPF & OpenXML4J

- For .doc, HWPF provides partial support
- For .docx, OpenXML4J provides partial support, but that is not being developed
- POI provides XWPF, which handles the basics, and is being developed



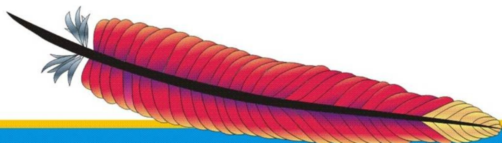
HWPF Overview

- .doc files are mostly concerned with runs of text
- File has ranges paragraphs, which have paragraphs, which have runs
- Formatting can be applied at any stage
- All text in a given run has the same formatting
- Many objects are actually stored as special text runs

ApacheCon
Leading the Wave
of Open Source

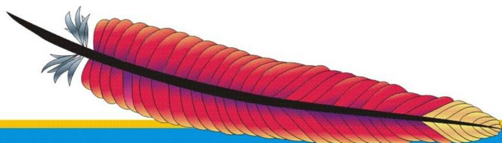
Supported Features

- Fetching, changing and inserting text
- Fetching and changing some text formatting (not all kinds supported)
- Extracting pictures



XWPF Overview

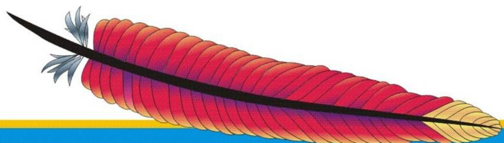
- .docx files are also mostly concerned with runs of text
- Runs can be in a default style, or the style info can be embeded inline
- The xml leaves a lot to be desired, but is fairly close to how .doc works
- Support is evolving



ApacheCon
Leading the Wave of Open Source

Apache POI

next section follows....



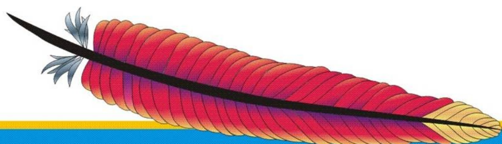
ApacheCon

Leading the Wave
of Open Source

Visio

- Visio can save as OLE2 or XML
- POI only works on the OLE2 .vsd format for now
- Despite being OLE2 based, .vsd has many differences to the other Office OLE2 file formats
- Code in scratchpad, still very limited support

ApacheCon

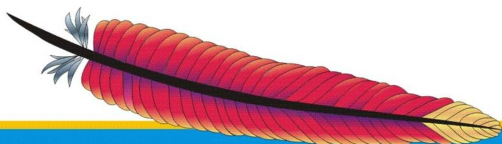


Leading the Wave
of Open Source

Streams, Chunks, Pointers

- Visio files are made up of lots of streams, which may be compressed
- Streams can hold pointers (to other streams), or chunks
- Chunks hold commands, such as text, drawings, page styles, colours etc
- Our knowledge of chunk commands comes from vsdump

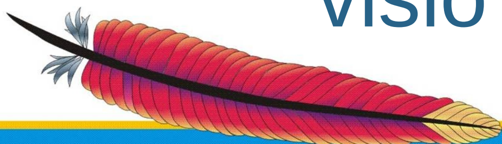
ApacheCon



Leading the Wave
of Open Source

vsdump

- http://www.gnome.ru/fileformats/diagrams.html
(site sometimes down though)
- Allows you to check the contents of a visio file
- Fairly good c based visio dumper, and limited python based visio dumper
- Source of lots of information on the visio file format

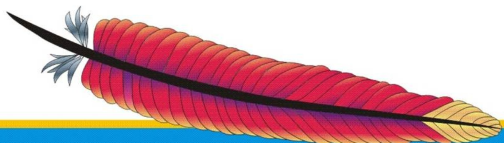


ApacheCon
Leading the Wave
of Open Source

ApacheCon

Apache POI

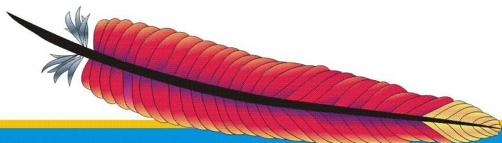
next section follows....



Leading the Wave
of Open Source

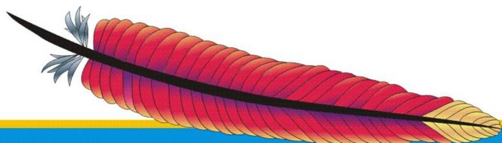
HPBF - Publisher

- Text extraction only for now
- We still don't know how much of the file format works
- One of the few formats for which Microsoft haven't released any docs
- Lots more reverse engineering is needed!



HSMF – Outlook MSG

- Mainly just text / message extraction
- File format makes heavy use of POIFS directory entries for different bits of the message
- Some entries are simple, and just contain the bits you want
- Some are more complex
- Hard bit is knowing which entry to get



ApacheCon
Leading the Wave
of Open Source