Excel Spreadsheets from RPG

With Apache's POI / HSSF

Presented by

Scott Klement

http://www.scottklement.com

© 2007-2012, Scott Klement

"There are 10 types of people in the world.
Those who understand binary, and those who don't."

Objectives Of This Session

- Learn when it makes sense to create spreadsheets with POI / HSSF.

- Learn how to create spreadsheets

- Learn how to modify existing spreadsheets

Oh, yeah… and provide lots of links to articles on the subject!

What is POI / HSSF?

- POI is a set of Java routines to work with (create, read, modify) Microsoft Office documents.

- Open source (free) software.

- Still in development.

- Created by the Jakarta Project, a project aimed at creating open source Java utilities. Jakarta has created many many utilities, including some that are very well known, such as Ant and Tomcat.

- HSSF is the component of POI that reads & writes Excel spreadsheets, it's not complete, but enough of it is available to do some really useful stuff.

  XSSF was added in 2009 to provide support for newer XML Excel format

- HWPF is the component that works with Microsoft Word. However, not enough of HWPF has been completed to make it useful.

- HSLF is the component that works with Microsoft Powerpoint files. Not enough of this one has been completed to make it useful.

3

Whaddya Mean, Java?!
I thought this presentation was about RPG?

- Yes, this presentation is about RPG – HSSF/XSSF can be used from an RPG program!

- Starting in V5R1, prototypes in RPG are able to call Java methods directly.

- Java methods are callable routines, very much like RPG subprocedures.

- That means that once you download and install POI, you can use the POI routines directly from an RPG program!

Of course, that also means:

- Have a basic understanding of how to use Java objects.

- Write prototypes and named constants that correctly access the APIs provided by XSSF/HSSF.

- But, I've done much of that for you, already!

4

Is This the Right Tool for Me?

- There are many other ways to create Excel spreadsheets:

  - CSV/TAB (CPYTOIMPF) = Easy, but has no formatting at all.

  - HTML table = Works, but has no capacity for formulas. Mapping browser functions to Excel can look weird.

  - XML = simple XML format only works with new Excel, has release compatibility problems, can be complex to code, especially with embedded pictures.

  - OOXML = (several XML docs embedded in a ZIP file, aka XLSX format) is supported by POI, but would be extremely complex to build yourself.

  - SYLK = Outdated, obscure.

    http://jplamontre.free.fr/jpltools.htm

  - BIFF (XLS format) = The one used by HSSF, and old Client Access file transfer. Excel's native & primary format up until Excel 2003 (but still works great in Excel 2007)

POI is Great For Reports

- POI works very nicely when you want to output an attractive report (instead of writing a spooled file or a PDF document.)

- Formulas are available, and that helps users interact with your report.

- Problem with POI is performance. It's slow.

- Works great for small reports, where bad performance isn't noticed.

- Or with long-running batch jobs, where the time it takes doesn't matter much.

A Little Java & OO Background

You don't need to understand the Java programming language to use HSSF, but it helps to understand a few of the concepts. I'll cover some of the basics here.

- What is a class?

- What is an object?

- How do you create an object?

- How can one object create another?

- What is the CLASSPATH?

- What is a JAR file?

- How does RPG support Java?

7

What is Object-Oriented?

Object-oriented languages are based on the concept of an object. The concept of an object comes from the real-world model of an object. If I were at home, I'd see many objects. My chair, my television, my computer, etc. Objects always have a "current state" and "behaviors".

Lets use dogs as an example. A dog has a current state:
- Hair is brown.
- Breed is collie.
- Location is kitchen.

And behaviors
- Dogs can bark
- Dogs can run

Note that behaviors can change the current state! (Running may change the location, for instance.)

Software objects are conceptually the same – the state is stored in "fields" (variables), and the behavior is carried out by calling a "method" (routine).

8

Classes (1 of 2)

Blueprint for an object. (e.g., a dog)

There are many dogs in the world, but they all fall into the same basic "class", (or "category") -- that of a dog.

(Kinda like a record format?)

Once you know the class, it can be used to create the individual dogs.


```java
Import java.lang;

Public class Dog {

    String color;
    String breed;
    int sex;    // 0=male, 1=female
    String location;

    public Dog(String c, String b,
               String p, int s) {
            color = c;
            breed = b;
         location = p;
              sex = s;
    }

    public void bark() {
        // insert code to make
        // barking noises
    }
```

Classes (2 of 2)

Fields, are variables that represent the current state of the object. You can think of these in the same way you think of fields in a data structure in RPG, or fields in a database record.

Methods, are like subprocedures (subroutines w/parameters) in RPG. They're little pieces of code that carry out a behavior.

Constructors are special methods that are called when an object is created. Sort of like *INZSR in RPG. (Except, they can receive parameters.)

```java
... Code continued from last slide ...

    public void eat(DogFood food) {
        // code to eat an object of
        // the DogFood class goes here
    }

    public void comeRunning(String l) {
        location = l;
    }

    public Dog havePuppy(Dog father){
        // code to mix attributes
        // of mother and father go
        // here.
    }

}
```

An object is an "instance" of a class. A class can't really be used by itself, instead it must be used to create an object. To create an object in Java, you use the "new" keyword. That creates a new object ("a dog") from a class ("blueprint for a dog").

You can pass parameters to the constructor when you create an object.

```
Dog mack = new Dog("brown","collie", "kitchen", 0);
Dog lady = new Dog("white", "mutt", "living room", 1);
```

Now that you have an object, you can access it's fields and call it's methods:

```
if ( mack.location == "kitchen" ) {
    DogFood df = new DogFood("alpo", "beef");
    mack.eat(df);
}
```

11

Sometimes, instead of a constructor, you create an object by calling a method that's in a different object. This is typically done when there's a close relationship between the objects.

```
if ( lady.sex != mack.sex ) {
    lady.bark();
    mack.comeRunning(lady.location);
    Dog rover = lady.havePuppy(mack);
}
```

In this example, the "rover" object might be created by taking some of the attributes from the lady object and some of the attributes of the mack object, so that you don't have to specify them all in a call to the constructor.

Note that we call the methods directly in the objects themselves, not in the class!

12

RPG's Support for Java

RPG supports calling Java methods. *(Starting in V5R1)*

RPG does not have direct support for accessing fields in a Java object or class. You have to call a Java method that returns the field, or call an API to retrieve the field. *(But most Java classes do not make fields available, anyway, as it's considered a bad practice.)*

Documented by IBM in the ILE RPG Programmer's Guide
Chapter 11 "RPG and the e-Business World"

Features added to support Java method calls are:
- **O** *data type in the D-spec.*
- **CLASS(*JAVA : 'class-name')** *D-spec keyword (used with O data type)*
- **EXTPROC(*JAVA : 'class-name' : 'method-name')** *on prototypes.*
- *Special value of* ***CONSTRUCTOR** *for 'method-name', above.*

13

Example Constructor Prototype

For example, to create a Java String object (which is how Java stores alphanumeric data), you'd have to create a prototype that calls the constructor for the java.lang.String class:

```
D new_String       pr          O    Class(*Java:'java.lang.String')
D                                    ExtProc(*Java
D                                    :'java.lang.String'
D                                    :*CONSTRUCTOR)
D   value                32767A    varying
```

This prototype:
- Returns a Java object.
- That object is to be an instance of the 'java.lang.String' class
- Java class names are case-sensitive. (string, String and strING are different)
- It creates an object (calls the *constructor).
- Passes a variable-length string as a parameter to the constructor.

14

Example Constructor Call

- To create a string, you call the prototype (shown on last screen)
- You need a "type O" field to receive the result.
- Simply declaring the type O field does not create a string – only a placeholder for one.
- The call to the *CONSTRUCTOR prototype is what actually creates the string.

```
D breed           s            O    Class(*Java:'java.lang.String')

        breed = new_String('collie');
```

Tip: Typing `CLASS(*JAVA:'java.lang.String')` repeatedly can be very tiring. (Same goes for any Java class name!) Here's an easier way:

```
D jString         s            O    Class(*Java:'java.lang.String')
. . .
D color           s                 like(jString)
D breed           s                 like(jString)
D place           s                 like(jString)
```

LIKE on Prototypes

LIKE can also be used on prototypes:

```
D Dog             s            O    class(*java:'Dog')

D new_Dog         pr                like(Dog)
D                                   ExtProc(*Java:'Dog':*CONSTRUCTOR)
D    color                          like(jString)
D    breed                          like(jString)
D    place                          like(jString)
D    sex                   10i 0 value
```

```
    D mack          s                 like(Dog)
    D color         s                 like(jString)
    D breed         s                 like(jString)
    D place         s                 like(jString)

      /free

          color = new_String('brown');
          breed = new_String('collie');
          place = new_String('kitchen');
          mack = new_Dog(color: breed: place: 0);
```

Obtaining JARs for POI

- Go to the Apache web site for POI:
  - *http://poi.apache.org*
- *Click the "download" link.*
- *Use the mirror they suggest.*
- *Click Release / Bin.*
- *All my code has been tested with poi-bin-3.6-20091214.zip*
  - *(Though, older versions exist for POI 3.1 and earlier!)*
- *Extract the JAR files to a directory in your IFS. (example: /poi)*

Add the JAR files to your CLASSPATH variable:

```
ADDENVVAR ENVVAR(CLASSPATH)
          VALUE('...IFS pathnames for JAR files...')
```

- Must be done before JVM is started / used in job.
- Must be re-done each time you sign on (unless you set it at the system-level.)

17

Obtaining HSSFR4 Service Program

As part of a series of articles that I wrote for the System iNetwork Programming Tips newsletter, I've written a lot of prototypes, constants, and helper subprocedures (in RPG) for working with POI.

The most up-to-date and well tested copy of my HSSF code can be found and downloaded from my we site:

http://www.scottklement.com/poi/

This site contains the downloads for HSSFR4 for POI versions 3.0-3.1 (without XLSX support) and 3.6 (with XLSX support)

It also contains links to all of the articles I've written about POI -- some of which have older versions of the code.

But please use the code from my site, as it's the latest version with the latest bug fixes.

18

You Also Need

Anytime you use Java from RPG on IBM i, you need the following licensed programs:

- `57xx-SS1, opt 13 System Openness Includes (library QSYSINC)`
- `57xx-JV1, *BASE  Developer Kit for Java`
- `57xx-JV1, opt 7  Java Developer Kit 5.0 ***`
- `57xx-WDS        ILE RPG compiler`

*** JDK 5.0 is the minimum version for POI 3.6. Version 1.4 worked fine for older versions. Each version has a different '5722-JV1, opt' number.

All of these licensed programs are shipped on the CDs with IBM i. The only one that's an extra charge is the ILE RPG compiler, and you already own that (most likely.)

Extra Java Components

To run version 3.6 with support for the XLSX file formats, you also need to install:

- `POI 3.6 (free from Apache)`
- `DOM4J 1.6.1 (free from Apache)`
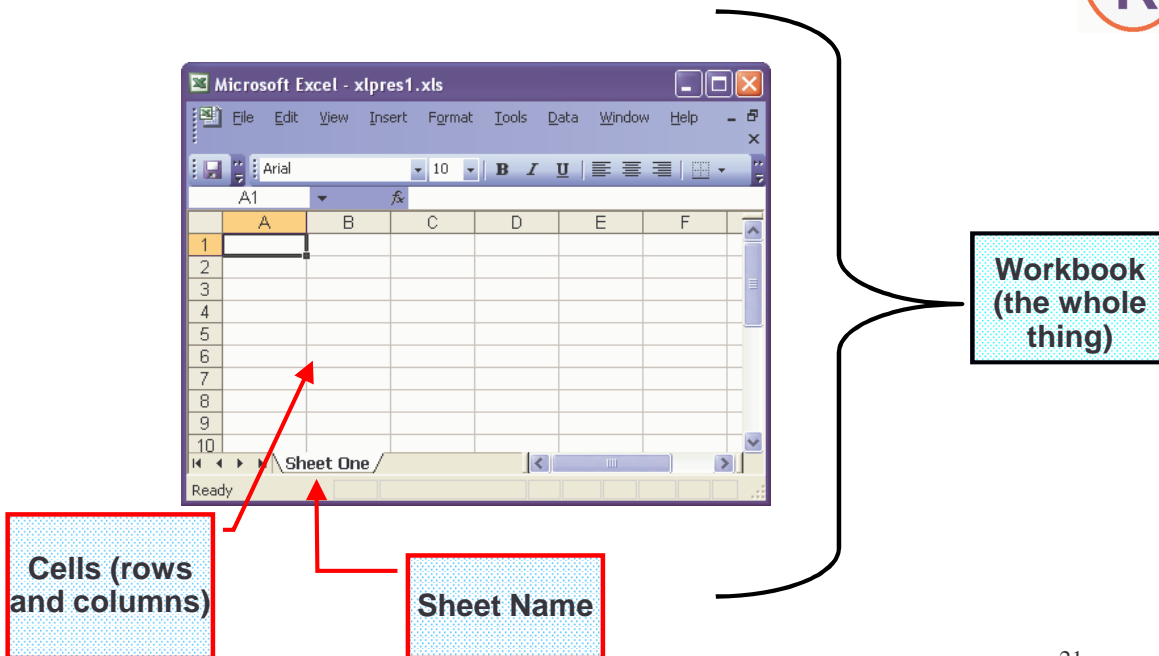- `XmlBeans 2.5.0 (free from Apache)`

These are not required for traditional XLS support. They are only needed for the newer XLSX support.

For detailed information on installing these, and how to set up environments to allow both version 3.1 and 3.6 on the same system, please see the following article:
http://systeminetwork.com/article/new-poi-release-brings-excels-xlsx-support-rpg

Creating a Spreadsheet (finally!)

Workbook (the whole thing)

Cells (rows and columns)

Sheet Name

Code to Create Empty Book/Sheet

The following code creates a blank Excel document in your System i computer's memory:

```
 /copy hssf_h

D Str             s                     like(jString)
D Sheet           s                     like(SSSheet)
D book            s                     like(SSWorkbook)

 /free

   book = new_HSSFWorkbook();    // or new_XSSFWorkbook() for xlsx

   Str = new_String('Sheet One');
   Sheet = SSWorkbook_createSheet(Book: Str);
```

- Prototypes and object types are defined in the HSSF_H copybook.
- The SSWorkbook object represents the "Excel document" as a whole.
  - Call new_HSSFWorkbook() to create traditional XLS format
  - Or call new_XSSFWorkbook() to create newer XLSX format.
- The SSSheet object represents a single sheet ("tab") in the Excel document.
- As soon as you use any Java call (including the new_HSSFworkbook call) the JVM will be loaded. You must set CLASSPATH before that.

Simplifying Strings

Each time you want to pass a character string to a Java object, you have to first create a String object with the character string inside it, then pass that. I find that to be cumbersome, so I put subprocedures in HSSFR4 that do that for me.

```
P ss_NewSheet      B                  EXPORT
D ss_NewSheet      PI                 like(SSSheet)
D   Book                              like(SSWorkbook)
D   Name                    1024A     const varying

D Str             s                   like(jString)
D Sheet           s                   like(SSSheet)
 /free
    Str = new_String(Name);
    Sheet = SSWorkbook_createSheet(Book: Str);
    SS_freeLocalRef(Str);
    return Sheet;
 /end-free
P                  E
```

Now I can simply do this:

```
Sheet = SS_newSheet(book: 'Sheet One');
```

Adding Rows and Cells (1 of 2)

```
D row             s                   like(SSRow)
D cell            s                   like(SSCell)

    Row = SSSheet_createRow(Sheet: 0);
    Cell = SSRow_createCell(Row: 0);

    Str = new_String('Hello World');
    SSCell_setCellValueStr(Cell: Str);

    Cell = SSRow_createCell(Row: 1);
    SSCell_SetCellValueD(Cell: 12345.60);

    SS_save(book: '/tmp/xlpres1.xls');
    *inlr = *on;
```

- SSSheet_createRow() asks the sheet object to create a new row.
  - ➢ POI numbers rows starting with 0
  - ➢ Row numbers are always one less than those in Excel

- SSRow_createCell() creates a new cell in the given row.
  - ➢ POI numbers columns (cells) starting with 0.   so 0=A, 1=B, 2=C, 3=D, etc.

- SSCell_setCellValueStr() sets the value of a cell to a character string.
- SSRow_setCellValueD() sets the value of a cell to a numeric value.
  - ➢ D stands for "Double-Precision Floating Point"

- SS_save() is an RPG subprocedure in HSSFR4 that saves the entire workbook (and all objects inside it) to an Excel spreadsheet in the IFS.
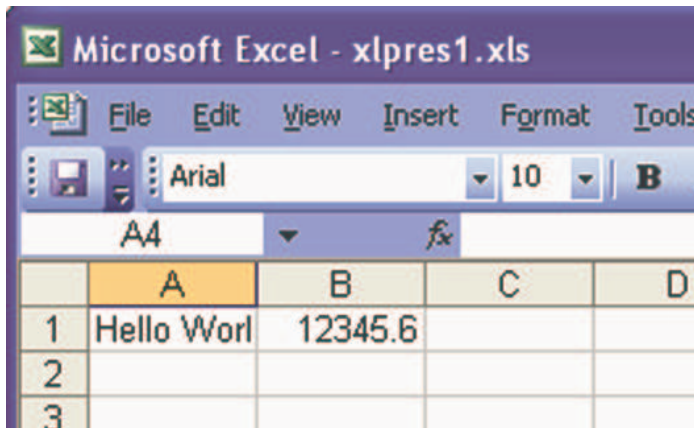
Note that:
- First column isn't large enough to fit "Hello World"
- Second column dropped trailing zeroes.

25

Column Widths

The Sheet object has a method named `setColumnWidth` that you can use to control the width of each column. The widths are in 1/256th of the size of the average character.

```
book = new_HSSFWorkbook();     // or new_XSSFWorkbook();
Sheet = SS_newSheet(book: 'Sheet One');

SSSheet_setColumnWidth( sheet: 0:  15 * 256 );
SSSheet_setColumnWidth( sheet: 1:  10 * 256 );
```

The preceding code sets the width
- Column A to 15 chars wide
- Column B to 10 chars wide

Since I'm using a proportional font, the above numbers are only approximate, so pick something sufficiently large.

26

A cell style is an object that contains information about how to format a cell.it has info about:
Font, Font Size, Color, Boldness, Italics, Underlines, Numeric Format, Currency, Alignment
(Alignment can be Left, Right or Centered)
*…and many other attributes as well… you can find details in the articles listed at the end of this
presentation….*

```
D Numeric         s                      like(SSCellStyle)
D DataFmt         s                      like(SSDataFormat)
D NumFmt          s               5I 0
   . . .
    Numeric = SSWorkbook_createCellStyle(book);

    DataFmt = SSWorkbook_createDataFormat(book);
    Str = new_String('#,##0.00');
    NumFmt = SSDataFormat_getFormat(DataFmt: Str);
    SSCellStyle_setDataFormat(Numeric: NumFmt);

    SSCellStyle_setAlignment(Numeric: ALIGN_RIGHT);
```

Cell styles are stored in an array inside the SSWorkbook object. Think of this array as a closet
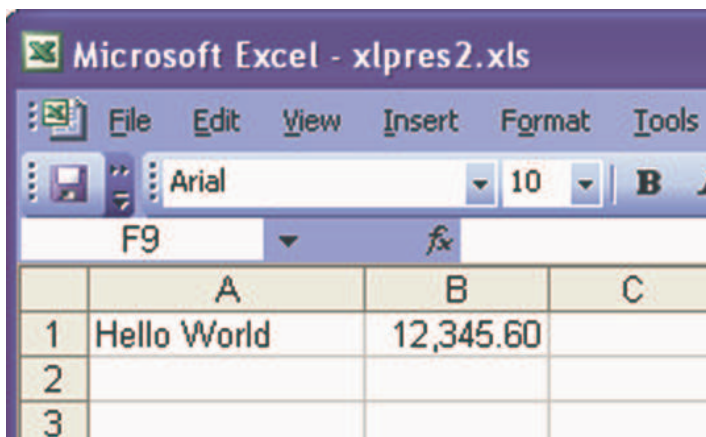full of clothes that your cells can wear to change the way they look!

Once you have your cell styles created in the workbook, you can apply them to individual cells
to "dress them up."

```
    Cell = SSRow_createCell(Row: 1);
    SSCell_SetCellValueD(Cell: 12345.60);

    SSCell_setCellStyle(Cell: Numeric);
```


You can apply the same cell
style (without re-creating it)
to as many cells as you like.

What I've shown you with
cell styles is just the tip of the
iceberg! You can perform
any type of formatting
imaginable.

Simplified Cell Creation

As you can see, for each cell you want to create, you have to createCell, new_String (sometimes), setCellValue, setCellStyle. freeLocalRef (more later).

Too cumbersome! I created shortcuts: ss_text(), ss_num(), ss_date(), ss_formula().

ss_xxxx(*row-obj* : *col-no* : *cell-value* : *cell-style*);

```
rowno = -1;
read ITEMLIST;

dow not %eof(ITEMLIST);

    rowno = rowno + 1;
    Row = SSSheet_createRow(Sheet: rowno);

    ss_text( Row: 0: Item_sku  : Normal  );
    ss_text( Row: 1: Item_Desc : Normal  );
    ss_num ( Row: 2: Price     : Numeric );
    ss_num ( Row: 3: StockQty  : Numeric );
    ss_date( Row: 4: LastBought: UsaDate );

    read ITEMLIST;
enddo;
```

Each procedure takes
- Row object
- Cell column number
- Value
- Cell Style

Value will be:
- **ss_text** = RPG alpha field
- **ss_num** = number field
- **ss_date** = date field.
- **ss_formula** = RPG alpha field containing an Excel formula. (next slide…)

29

Inserting a Formula

ss_formula(*row-obj* : *col-no* : *formula-text* : *cell-style*);

My shortcut routine ss_formula() lets you insert a formula into an Excel spreadsheet.
- But cells must be specified in Excel's notation, such as A3 or B29.
- You do **not** put an equal sign in front of a formula, like you would in Excel.
- I created a utility routine to calculate Excel's cell name:

excel-cell-name = ss_cellname(*hssf-row-no* : *hssf-col-no*);

```
...this would follow the code from the last slide ...

start = SS_cellName(     0: 3);    // Get cell name (D1)
end   = SS_cellname(rowno: 3);     // Get cell name (Dx)

rowno = rowno + 2;
Row = SSSheet_createRow(Sheet: rowno);
ss_formula(Row: 3: 'SUM('+start+':'+end+')': Numeric );
```

30

Garbage Collection

All of the Java objects that make up the spreadsheet are loaded into the computers memory.

They are *not* automatically removed from memory.

- Not when workbook is saved.
- Not when program ends (even with LR on)
- Not when you run RCLRSC
- Not even when the activation group ends!

The JVM knows when Java is done with objects, so for Java they're automatically "garbage collected" (cleaned up when nothing still uses the object.)

The JVM does not know when RPG is done with them, because JVM's are designed to understand how RPG works! (They are part of the Java language.)

You have to tell the JVM when you're done with each object by calling an API.

ss_FreeLocalRef

The *ILE RPG Programmer's Guide* has sample code that most people use to notify the JVM when their RPG programs are done with objects.

I have made my own versions, based on IBM's sample code, that are included in HSSFR4 along with the other utilities. You can call them from your programs to clean up your objects.

ss_freeLocalRef(*object-name*) is one way to tell the JVM you're done with an object. It frees up one object at a time.

```
 D Str              s                  like(jString)


     Str = new_String('Hello World');
     SSCell_setCellValueStr(Cell: Str);


     ss_freeLocalRef(Str);    // done with Str
```

Freeing Objects in Groups

The other way to free objects is to create an object group. The following code creates an object group with space for 10000 objects:

```
ss_begin_object_group(10000);
```

From this point, every object created (including those created inside the Java routines that you call) will be placed inside the group.

Think of the group as a cardboard box. Every object you create will exist inside that box.

When you want to free them up, you simply discard the whole box.

```
ss_end_object_group();
```

You can create sub-groups inside other groups as well.

Tip: Always start a group when your program starts, and end it when the program ends, and you'll never have extra Java objects trapped in memory.

33

Object Group Example

```
ss_begin_object_group(10000);

    Row = SSSheet_createRow(Sheet: 0);
    Cell = SSRow_createCell(Row: 0);

    Cell = SSRow_createCell(Row: 1);
    SSCell_SetCellValueD(Cell: 12345.60);

    . . . Lots of other code can be here . . .

ss_end_object_group();
```

All objects after the "begin" are cleaned up when the "end" is called.

34

Reading a Spreadsheet

There are two ways of reading a spreadsheet.

1. Event API.
   - HSSF parses the entire workbook.
   - Each time a cell with a value is found, it's considered an "event".
   - HSSF calls RPG subprocedures (that you write) with each cell value found.
   - Runs fast, very simple to code, but does not allow updates.
   - Requires my XLPARSE service program & Java classes from
     http://www.scottklement.com/poi

2. User Model API
   - You load workbook into Java objects in memory.
   - You call routines like getSheet(), getRow() and getCell() to read each cell individually
   - The workbook is loaded into memory just like the ones you create, so you can also use the createSheet(), createRow(), setCellValue(), etc. to change existing values or add new ones to existing sheet.
   - Runs slower, requires more work to code, but is more versatile.

Event API

Tell HSSF the spreadsheet to parse, and which subprocedures to call for numeric cells and character cells.

```
    callp xlparse_workbook('/usr/reports/InventoryList.xls'
                    : %paddr( NumericCell )
                    : %paddr( CharCell    ) );
```

you write the subprocedures that HSSF calls for each cell as follows…

```
P CharCell      B
D CharCell      PI            1N
D   Sheet              1024A   varying
D                              const
D   Row              10I 0 value
D   Column            5I 0 value
D   Value            32767A   varying
D                              const
 /free
    if (row>=0 and row<=22);
       select;
       when col = 0;
          sku = value;
       when col = 1;
          desc = value;
 ...etc...
```

```
P NumericCell   B
D NumericCell   PI            1N
D   Sheet              1024A   varying
D                              const
D   Row              10I 0 value
D   Column            5I 0 value
D   Value             8F    value
 /free
    if (row>=0 and row<=22);
       select;
       when col = 2;
          eval(h) price = value;
       when col = 3;
          eval(h) qty   = value;
 ...etc...
```

User Model API

SS_open() loads an existing sheet into memory. You can now get the Java objects from it and read their values, change them, whatever you like…

```
D book            s                   like(SSWorkbook)
D sheet           s                   like(SSSheet)
D row             s                   like(SSRow)
D cell            s                   like(SSCell)

  book = ss_open('/usr/reports/InventoryList.xls');
  sheet = ss_getSheet(book: 'Sheet One');
  row  = SSSheet_getRow(sheet: 7);
  cell = SSRow_GetCell(row: 2);
  type = SSCell_getCellType(cell);

  StrVal = 'Cell C8 = ';

  select;
  when type = CELL_TYPE_STRING;
      StrVal += String_getBytes(SSCell_getStringCellValue(cell));
  when type = CELL_TYPE_FORMULA;
      StrVal += String_getBytes(SSCell_getCellFormula(cell));
  when type = CELL_TYPE_NUMERIC;
      NumVal = SSCell_getNumericCellValue(cell);
      StrVal += %char(%dech(NumVal:15:2));
  endsl;

  dsply StrVal;
```

37

More Information

Apache's main Web site for POI:
http://poi.apache.org

- **Downloads.**
- **Javadocs ("reference manual" style documentation for all POI routines)**
- **Click HSSF for HSSF tutorials (intended for Java programmers)**

IBM's (Websphere Development Studio) ILE RPG Programmer's Guide
(part of Information Center)
See "Chapter 11: RPG and the e-Business World"

38

Scott's HSSF Articles

Scott has written 18 articles about HSSF from RPG

- A few introductory articles explain the basics.

- Many very small articles demonstrate one particular feature. (e.g. How to change the paper orientation, how to reference another Excel document, how to enable word wrapping, etc).

List of all articles is available on Scott's web site:

    http://www.scottklement.com/poi/

This Presentation

You can download a PDF copy of this presentation from:

http://www.scottklement.com/presentations/

Thank you!