POI Ruby Bindings

by Avik Sengupta

1. Intro

The POI library can now be compiled as a Ruby extension, allowing the API to be called from Ruby language programs. Ruby users can therefore read and write OLE2 documents, such as Excel files with ease

The bindings are generated by compiling POI with gcj, and generating the Ruby wrapper using SWIG. The aim is the keep the POI api as-is. However, where java standard library objects are used, an effort is made to transform them smoothly into Ruby objects. Therefore, where the POI API takes an OutputStream, you can pass an IO object. Where the POI works java.util.Date or java.util.Calendar object, you can work with a Ruby Time object.

2. Getting Started

2.1. Pre-Requisites

The bindings have been developed with GCC 3.4.3 and Ruby 1.8.2. You are unlikely to get correct results with versions of GCC prior to 3.4 or versions of Ruby prior to 1.8. To compile the Ruby extension, you must have GCC (compiled with java language support), Ruby development headers, and SWIG. To run, you will need Ruby (obviously!) and *libgcj* , presumably from the same version of GCC with which you compiled.

2.2. Subversion

The POI-Ruby module sits under the POI Subversion. Running *make* inside that directory will create a loadable ruby extention *poi4r.so* in the release subdirectory. Tests are in the *tests/* subdirectory, and should be run from the *poi-ruby* directory. Please read the tests to figure out the usage.

Note that the makefile, though designed to work accross Linux/OS X/Cygwin, has been tested only on linux. There are likely to be issues on other platform; fixes gratefully accepted!

2.3. Binary

A version of poi4r.so is available here. Its been compiled on a linux box with GCC 3.4.3 and Ruby 1.8.2. It dynamically links to libgcj. No guarantees about working on any other box.

3. Usage

The following ruby code shows some of the things you can do with POI in Ruby

```
h=Poi4r::HSSFWorkbook.new
#Test Sheet Creation
s=h.createSheet("Sheet1")

#Test setting cell values
s=h.getSheetAt(0)
r=s.createRow(0)
c=r.createCell(0)
c.setCellValue(1.5)

c=r.createCell(1)
c.setCellValue("Ruby")

#Test styles
st = h.createCellStyle()
c=r.createCell(2)
st.setAlignment(Poi4r::HSSFCellStyle.ALIGN_CENTER)
c.setCellStyle(st)
c.setCellValue("centr'd")

#Date handling
c=r.createCell(3)
t1=Time.now
c.setCellValue(Time.now)
t2= c.getDateCellValue().gmtime

st=h.createCellStyle();
st.setDataFormat(Poi4r::HSSFDataFormat.getBuiltinFormat("m/d/yy h:mm"))
c.setCellStyle(st)

#Formulas
c=r.createCell(4)
c.setCellFormula("A1*2")
c.getCellFormula()

#Writing
h.write(File.new("test.xls","w"))
```

The *tc_base_tests.rb* file in the *tests* sub directory of the source distribution contains examples of simple uses of the API. The quick quide is the best place to learn HSSF API use.

(Note however that none of the Drawing features are implemented in the Ruby binding.) See also the POI API documentation for more details.

4. Future Directions

4.1. TODO's

- Implement support for reading Excel files (easy)
- Expose POIFS API to read raw OLE2 files from Ruby
- Expose HPSF API to read property streams
- Tests... Tests... Tests...

4.2. Limitations

- Check operations in 64bit machines - Java primitive types are fixed irrespective of machine type, unlike C/C++ types. The wrapping code that converts C/C++ primitive types to/from Java types is making assumptions on type sizes that MAY be incorrect on wide architectures.
- The current implementation is with the POI 2.0 release. The 2.5 release adds support for Excel drawing primitives, and thus has a dependency on java AWT. Since AWT is not very mature in gcj, leaving it out seemed to be the safer option.
- Packaging - The current make file makes no effort to install the extension into the standard ruby directories. This should probably be packaged as a gem.