# POI 1.0 Vision Document

**by Andrew C. Oliver, Marcus W. Johnson**

### 1. Preface

(21-Jan-02) While this document is just full of useful project introductory information and I do suggest those interested in getting involved in the project read it, it is woefully out of date.

We deliberately allowed this document to run out of date because it is a good reflection of what the original vision was for POI 1.0. You'll note that some of the terminology is not used in quite the same way any longer. I've made some minor corrections where reading this confused me. An example: in some places this document may refer to POI API instead of POIFS API. When this vision was written we had an incomplete understanding of the project.

Lastly, the scope of the project expanded dramatically near the end of the 1.0 cycle. Our vision at the time was to focus merely on the Excel port (having no idea how the project would grow or be received) and provide the OLE 2 Compound Document port for others to port later formats. We now plan to spearhead these ports under the umbrella of the POI project. So, you've been warned. Read on, but just realize that we had a fuzzy view of things to come, and hindsight is 20-20.

If I recall major holes were: a complete understanding of the format of OLE 2 Compound Document format, Excel file format, and exactly how Cocoon 2 Serializers worked. (that just about covers the whole range huh?)

### 2. 1. Introduction

## 2.1. 1.1 Purpose of this document

The purpose of this document is to collect, analyze and define high-level requirements, user needs and features of the HSSF Serializer for Cocoon 2 and related libraries. The HSSF Serializer is a java class supporting the Serializer interface from the Cocoon 2 project and outputting in a compatible format of that used by the spreadsheet program Microsoft Excel '97. The HSSF Serializer will be responsible for converting XML spreadsheet-like documents into Excel-compatible XLS spreadsheets.

## 2.2. 1.2 Project Overview

Many web apps today hit a brick wall when it comes to the user request that they be able to easily manipulate their reports and data extracts in the popular Microsoft Excel spreadsheet format. This often causes inferior technologies to be chosen for the project simply because they easily support this format. This project seeks to extend existing XML, Java and Apache Cocoon 2 project technologies by:

- providing an extensible library (POIFS) which reads/writes in a compatable format to OLE 2 Compound Document Format (aka Structured Storage Format) for easy implementation of other document types;
- providing a library (HSSF) for manipulating spreadsheet data and outputting it in a compatible format to Microsoft Excel XLS format;
- and providing a Cocoon 2 Serializer (HSSFSerializer) for serializing XML documents as Excel-compatible spreadsheets.

**3. 2. User Description**

## 3.1. 2.1 User/Market Demographics

There are a number of enthusiastic users of XML, UNIX and Java technology. Secondly, the Microsoft solution for outputting Office Document formats often involves actually manipulating the software as an OLE Server. This method provides extremely low performance, extremely high overhead and is only capable of handling one document at a time.

1. Our intended audience for the HSSF Serializer portion of this project are developers writing reports or data extracts in XML format.
2. Our intended audience for the HSSF library portion of this project is ourselves as we are developing the Serializer and anyone who needs to write to Excel spreadsheets in a non-XML Java environment or who has specific needs not addressed by the Serializer.
3. Our intended audience for the "POIFS" OLE 2 Compound Document format reader/writer is ourselves as we are writing the HSSF library and secondly, anyone wishing to provide other libraries for reading/writing OLE 2 Compound Document Format in Java.

## 3.2. 2.2. User environment

The users of this software shall be developers in a Java environment on any Operating System or power users who are capable of XML document generation/deployment.

## 3.3. 2.3. Key User Needs

The OLE 2 Compound Document format is undocumented for all practical purposes and cryptic for all impractical purposes. Developer needs in this area include documentation and an easy to use library for reading and writing in this format without requiring the developer to have intimate knowledge of the format.

There is currently no good way to write to Microsoft Excel documents from Java or from a non-Microsoft Windows based platform for that matter. Developers need an easy to use library that supports a reasonable feature set and allows seperation of data from formatting/stylistic concerns.

There is currently no good way to transform XML data to Microsoft Excel. Apache's Cocoon 2 project supplies a complete framework for XML, but nothing for outputting in Excel's XLS format. Developers and power users alike need a simple method to output XML documents to Excel through server-side processing.

## 3.4. 2.4. Alternatives and Competition

Originally there weren't any decent alternatives for reading or writing to Excel. This has changed somewhat.

### 4. 3. Project Overview

## 4.1. 3.1. Project Perspective

The produced code shall be licensed by the Apache License as used by the Cocoon 2 project and maintained on a project page until such time as the Cocoon 2 developers accept it as a donation (at which time the copyright will be turned over to them).

## 4.2. 3.2. Project Position Statement

For developers on a Java and/or XML environment this project will provide all the tools necessary for outputting XML data in the Microsoft Excel format. This project seeks to make the use of Microsoft Windows based servers unnecessary for file format considerations and to fully document the OLE 2 Compound Document format. The project aims not only to provide the tools for serializing XML to Excel's file format and the tools for writing to that file format from Java, but also to provide the tools for later projects to convert other OLE 2 Compound Document formats to pure Java APIs.

## 4.3. 3.3. Summary of Capabilities

HSSF Serializer for Apache Cocoon 2

| Benefit | Supporting Features |
|---|---|
| Standard XML tag language for sheet data | Serializer will transform documents utilizing a defined tag language |
| Utilize XML to output in Excel | Serializer will output in Excel |
| Java API to output in Excel on any platform | The project will develop an API that outputs in Excel using pure Java. |
| Make it easy for developers to port other OLE 2 Compound Document-based formats to Java. | The POIFS library will contain both a high-level abstraction along with low-level constructs. The project will fully document the OLE 2 Compound Document Format. |

## 4.4. 3.4. Assumptions and Dependencies

- The HSSF Serializer will run on any Java 2 supporting platform with Apache Cocoon 2 installed along with the HSSF and POIFS APIs.
- The HSSF API requires a Java 2 implementation and the POI API.
- The POIFS API requires a Java 2 implementation.

### 5. 4. Project Features

The POIFS API will include:

- Low level structures representing the structures in a POI filesystems.
- A low-level API for creating/manipulating POI filesystems.
- A set of high level interfaces abstracting the user from the POI filesystem constructs and representing it as a standard filesystem (Files, directories, etc)

The HSSF API will include:

- Low level structures representing the structures in an Excel file.
- A low-level API for creating and manipulating Excel files and writing them into POI filesystems.
- A high level model and style interface for manipulating spreadsheet data without knowing anything about the Excel format itself.

## 5.1. 4.1 POI Filesystem API

The POI Filesystem API includes:

- An implementation of Big Blocks
- An implementation of Small Blocks
- An implementation of Header Blocks
- An implementation of Block Allocation Tables
- An implementation of Property Sets
- An implementation of the POI filesystem including functions to get and set the above constructs; compound functions for reading/writing files/directories.
- An abstraction of the POI filesystem providing interfaces representing Files, Directories, FileSystems in normal terminology and encapulating the above constructs.
- Full documentation of the POI file format.
- Full documentation of the APIs and interfaces provided through Javadoc, user documentation (aimed at developers using the APIs)
- Examples aimed at teaching the user to write code using POI. (titled: recipes for POI)
- Performance specifications. (Example POI filesystems rated by some measure of complexity along with system specifications and execution times for given operations)

## 5.2. 4.2 HSSF API

The HSSF API includes:

- An implementation of Record (binary 2 byte type followed by 2 byte size (n) followed by n bytes)
- Implementations of many standard record types mapping the data bytes to fields along with methods to reserialize those fields
- An implementation of the HSSF File including functions to get/set the above constructs, create a blank file with the minimum required record types and mappings between getting/setting data and style in a workbook to the creation of record types, and read HSSF files.
- An abstraction of the HSSF file format providing interfaces representing the HSSF File, HSSF Workbook, HSSF Sheet, HSSF Column, HSSF Formulas in a manner seperating the data from the styling and encapsulating the above constructs.
- Full documentation of the HSSF file format (which will be a subset of the Excel '97 File format). This must be done with care for legal reasons.
- Full documentation of the APIs and interfaces provided through Javadoc, user documentation (aimed at developers using the apis).
- Examples aimed at teaching developers to use the APIs.
- Performance specifications. (Example files rated by some measure of complexity along with system specifications and execution times for given operations - possibly the same files used for POI's tests)

## 5.3. 4.3 HSSF Serializer

The HSSF Serializer subproject:

- A class supporting the Cocoon 2 Serializer Interface.
- An interface between the SAX events and the HSSF APIs.
- A specified tag language for using with the Serializer.
- Documentation on the tag language for the HSSF Serializer
- Normal javadocs.
- Example XML files
- Performance specifications. (Example XML docs and stylesheets rated by some measure of complexity along with system specifications and execution times)

**6. 5. Other Product Requirements**

## 6.1. 5.1. Applicable Standards

All Java code will be 100% pure Java.

## 6.2. 5.2. System Requirements

The minimum system requirements for POIFS are:

- 64 Mbytes memory
- Java 2 environment
- Pentium or better processor (or equivalent on other platforms)

The minimum system requirements for HSSF are:

- 64 Mbytes memory
- Java 2 environment
- Pentium or better processor (or equivalent on other platforms)
- POIFS API

The minimum system requirements for the HSSF Serializer are:

- 64 Mbytes memory
- Java 2 environment
- Pentium or better processor (or equivalent on other platforms)
- Cocoon 2
- HSSF API
- POI API

## 6.3. 5.3. Performance Requirements

All components must perform well enough to be practical for use in a webserver environment (especially Cocoon2/Tomcat/Apache combo)

## 6.4. 5.4. Environmental Requirements

The software will run primarily in developer environments. We should make some allowances for not-highly-technical users to write XML documents for the HSSF Serializer. All other components will assume intermediate Java 2 knowledge. No XML knowledge will be required except for using the HSSF Serializer. As much documentation as is practical shall be required for all components as XML is relatively new, and the concepts introduced for writing spreadsheets and to POI filesystems will be brand new to Java and many Java developers.

### 7. 6. Documentation Requirements

## 7.1. 6.1 POI Filesystem

The filesystem as read and written by POI shall be fully documented and explained so that the average Java developer can understand it.

## 7.2. 6.2. POI API

The POI API will be fully documented through Javadoc. A walkthrough of using the high level POI API shall be provided. No documentation outside of the Javadoc shall be provided for the low-level POI APIs.

## 7.3. 6.3. HSSF File Format

The HSSF File Format as implemented by the HSSF API will be fully documented. No documentation will be provided for features that are not supported by HSSF API that are supported by the Excel 97 File Format. Care will be taken not to infringe on any "legal stuff".

## 7.4. 6.4. HSSF API

The HSSF API will be documented by javadoc. A walkthrough of using the high level HSSF API shall be provided. No documentation outside of the Javadoc shall be provided for the low level HSSF APIs.

## 7.5. 6.5. HSSF Serializer

The HSSF Serializer will be documented by javadoc.

## 7.6. 6.6 HSSF Serializer Tag language

The XML tag language along with function and usage shall be fully documented. Examples will be provided as well.

### 8. 7. Terminology

## 8.1. 7.1 Filesystem

filesystem shall refer only to the POI formatted archive.

## 8.2. 7.2 File

file shall refer to the embedded data stream within a POI filesystem. This will be the actual embedded document.