Technical White Paper

ACTUATE

Five Biggest Blunders when Building Spreadsheet Applications in Java

And how to avoid them

Table of Contents

The challenge of spreadsheets in Java

Java developers are asked to incorporate spreadsheets in Java to satisfy a number of different business needs. One common request of the Java developer is to build an application that will automatically generate and deliver Excel files. Another is to build an application that taps into the logic in existing spreadsheets, runs the calculations, and makes the results available to other applications or users.

While each of these presents the Java developer with a unique set of technical challenges, a common knowledge base about spreadsheets in Java creates a foundation for success across the application scenarios.

Without this knowledge base a Java developer can fall into traps that can be quite painful. This paper explores the 5 biggest blunders when building spreadsheet applications in Java. We hope it is educational and serves as a guide that helps Java developers build better spreadsheet applications.

Blunder #1:

Assume that any user will be happy with a given spreadsheet for long

Users expect to manipulate Excel files

Business employees are accustomed to using Excel to manipulate data to reveal the information they need and answer their "what if" scenarios. As such, when a Java developer is asked to automatically generate Excel reports, it is important that the user's expectation to manipulate the file be factored into the requirements. To meet this requirement, the Excel file delivered will need to support all the Excel features users are accustomed to – basic features such as formulas and charts and also the more advanced features such as pivot tables, auto filters and outlines that have now become commonplace among business users.

Business is fluid and so are Excel files

It is important to also understand that different people use the same data differently. To use revenue data as an example, one person may want to see the data broken out by region, another by type of sale (OEM vs. end user, channel partner vs. direct), another by vertical industry, another by product. And, even more importantly, over time a person's needs often change. Along the normal course of business, regions are split or combined, sales reps leave and new ones join, priorities shift and other changes take place that affect the information a user needs to see. This normal changing nature of business creates an ongoing stream of requests for new and modified reports. In addition, the value of the data grows as users have time to be with it and envision new possibilities. Paradoxically, the more successful your application, the more requests for changes.

Unfortunately, the longer it takes to deliver new or modified reports, the more the friction grows between end users and the application owner. Users want changes to be instantaneous; any lag time is perceived as a barrier to their success.

So, the reality is there will be a steady stream of ongoing requests for new and modified reports. It is important for the Java developer to consider the implications of this up front. The requirement is to architect the application in a way that allows for new and modified

report designs to be delivered extremely quickly. But, adding to this, if a changing a report design requires changing the Java code, the Java developer will always be the bottleneck and the speed of change will never satisfy users. So the requirement must also include the ability for the developer to share the report modification load with a broader set of users, which ideally includes non-technical users. The full requirement is to enable non-technical users to deliver new and modified reports extremely quickly, without modifying code.

Report automation requirements:

- Support all Excel functions
- The ability for non-technical users to deliver new and modified reports extremely quickly without writing code

Blunder #2:

Believe that the spreadsheet model will remain as it is

The spreadsheet model will change

The constantly changing nature of Excel files is also a major consideration in cases where the Java developer is building an application that taps into an existing spreadsheet. It is important that the Java developer give careful consideration to the fact that the spreadsheet model will change over time. The rate of change will vary from company to company but the fact of change will not. For some applications, the Excel model can change as frequently as every day. What are the implications of this to the project requirements? It means the application needs to operate independently of the spreadsheet. It needs to tap directly into the spreadsheet, run the calculations it finds in that spreadsheet, without replicating the spreadsheet.

If the application is built to replicate the spreadsheet model, the Java developer will need to learn the business rules and how they got applied into the model, but moreover, every time the spreadsheet model changes the developer will have to understand and replicate those changes. To prevent this scenario, the Java application must avoid duplicating the spreadsheet model or compiling the spreadsheet. The spreadsheet owner must be able to change the spreadsheet as he or she sees fit without “breaking” the Java application or creating work for the Java developer. The Java application must automatically accommodate changes to the spreadsheet model without developer intervention. And, in order to do this, it must be able to interpret all Excel functions that could be used in the spreadsheet, which is to say, it must be able to interpret all of Excel’s worksheet functions.

Calculation engine requirements:

- Automatically accommodate changes to the spreadsheet model without coding or manual intervention
- Be able to interpret all of Excel’s worksheet functions

Blunder #3:

Underestimate the need for data layouts and formatting

Layouts requested by users or dictated by the data are virtually unlimited

When the Java developer is building an application to generate Excel files, he or she is often given a specific, contained set of Excel files to deliver. It is critical for the developer to understand that this limited set of Excel files does not represent all the data layouts that will be requested by users or dictated by the data, which are virtually unlimited.

Some commonly requested layouts include:

- Layouts with parallel tables, such as used in budget vs. actual reports.
- Layouts that require inter-row and inter-column calculated values and sub-groupings that need to appear in a specified order (e.g. non-alphabetic), such as used for financial statements.
- Three-dimensional multi-sheet crosstab workbooks with cross-sheet references like a table of contents or a summary worksheet that subtotals the values on other worksheets, such as used for revenue reports broken down by channel, region, product, quarter, etc.
- Grid-style reports with “irregular” formatting where selected cells are formatted differently than others such as used in trend reports or any report that wants to highlight the best selling product, highest producing sales rep, etc.
- Reports with drill-to-detail using Excel’s ability to expand-to-detail and collapse-to-summary; reports that contain Visual Basic code, used for Excel Macros; reports that use 2D and 3D charts; pivot tables; data from multiple hierarchical data sources, text cells, merged cells, rotated text, hyperlinks, etc.

As such, when a Java developer sets out to generate Excel reports, it is important that the flexibility to accommodate any data layout or formatting be factored into his or her requirements.

Report automation requirement:

- Flexibility to accommodate any data layout or formatting

Blunder #4:

Underestimate the fluid nature of data

Either the data drives the layout or you do

When the Java developer is building an application to generate Excel files, he or she is often starting with a specific set of Excel files that show data for a specified time period, such as a report for the current year.

In order to build an application that can consistently generate clean reports without the need for constant manual intervention, the developer must factor into the application requirements the fluid nature of data. Quarters come and go, products will be added to product lines, the component mix for a product will change, geographies will some day relate differently to territories and territories differently to regions. This constant change is the nature of business and of time passing. Because of the constant change in business data, the application needs to automatically accommodate the data it finds. Stated differently, the application needs to dynamically render the report at run-time based on the data. Any layout that expects to be populated in a fixed way is a layout that will break as the data changes.

A yearly report run in June will need to populate six months worth of data grouped into two quarters, while that same report run in September will need to populate nine months worth of data grouped into three quarters. A rolling 12 month report will need to populate the report with the previous 12 months of data, relative to the month the report is run. A report that splits territories into regions and regions onto separate tabs should create the right number of tabs to correspond to what it finds in the data vs. being fixed for a certain number of regions. The data should dictate the rows, columns, and worksheets that appear in the report. The report should dynamically expand and contract as dictated by the data; columns should expand across the page, rows down the page, worksheets across tabs. The report layout should never need to include empty “placeholder” rows or columns, and should never require manual intervention to accommodate the passing of time or other changes that are part of normal business.

Report automation requirement:

- Reports should be automatically rendered based on what is in the data at run-time

Blunder #5:

Assume you will be able to move onto the next project without planning your exit

Avoid paths that create ongoing maintenance

Java developers who want to move onto other projects once their spreadsheet application is built need to carefully plan the project so they can extract themselves from it. Any piece of the project that has the potential to create ongoing maintenance should be given consideration in this exit planning.

For an application that taps into an existing Excel file and delivers the information and calculations it finds there to other applications or users, the potential ongoing maintenance piece is that of accommodating changes to the Excel model. As outlined in #2 above, this can be mitigated by employing a tool that automatically accommodates changes to the spreadsheet model. This eliminates any hard-coded interpretation of the business logic, thereby eliminating the ongoing maintenance needed to keep a hard-coded or compiled interpretation aligned with changes to the model. This approach allows the Java developer to create a great application without creating an ongoing labor burden for him or herself. In order for the application to successfully read Excel files and interpret Excel logic directly, it must also be able to interpret all the Excel functions that it might find when it taps into the spreadsheet model.

For an application that automates the delivery of Excel files to users, there will be ongoing requirements for new and modified Excel files (as outlined in #1). There are several ways the developer can mitigate this ongoing maintenance piece, including:

- Make it easy for users to personalize their own reports, allowing them to both filter the data via prompts and specify the structure they want for the spreadsheet (for example, chart or matrix format, single worksheet or cross-tab workbook). Use a deployment methodology that allows users to save their prompt values as a new report design, thereby creating new permutations of reports on their own without technical support or intervention.
- Use a toolset that comes with a fast and easy report designer for non-developers. This gives the Java developer the option of offloading the ongoing report modification responsibility to someone else.

Calculation engine requirements:

- Reads Excel files and interprets Excel logic directly. For example, can automatically pick up spreadsheet model changes on a network share without changing a single line of code
- Be able to interpret any worksheet function that might be used in the Excel model

Report automation requirement:

- A product that provides the option of a true Excel authoring tool for non-developers

About Actuate e.Spreadsheet Engine

Actuate e.Spreadsheet Engine is a pure Java toolset that can read, write and calculate the formulas within Excel files. It is the most complete Excel API for Java. As a pure Java toolset, e.Spreadsheet Engine works entirely inside the Java environment without any calls out to an external machine for processing. Because all of the processing takes place within the application, developers have complete control over the results and automatically inherit the security, scalability, and load tolerance features available with their J2EE application server. Unlike hand-coding Excel logic, or compiling spreadsheets into Java code, e.Spreadsheet Engine reads Excel files and processes Excel formulas directly, insulating the developer from having to learn and translate the Excel logic and maintain a replicated copy of that logic. Because the Java application supports all of Excel's worksheet functions, the business user is free to create, maintain and change the spreadsheet model according to their needs. e.Spreadsheet Engine has been embedded into thousands of applications across virtually all industries, including banking, insurance, finance, energy, pharmaceuticals and securities.

About Actuate e.Spreadsheet Designer

Actuate e.Spreadsheet Designer provides a visual report development environment for non-developers to quickly build full-featured Excel report templates without coding. Purpose-built as a true Excel report designer, it provides a host of powerful report development features such as:

- A built-in understanding of date relationships so that report data can be grouped automatically into months and quarters.
- The ability to easily set up dynamic cascading user prompts that filter the data or affect the structure of the spreadsheet that is rendered. For example, a prompt that asks the user for State, then presents the appropriate City selections based on the user's answer. The prompt screen might also ask the users for viewing preferences, for example, if they want to view the data in a chart or matrix format, or if they want the data displayed on just one worksheet or divided into many.
- A dashboard that provides a quick overview of data sources, data sets, parameters and security settings, as well as easy access to definitions for each scripting macro.

With a powerful package of usability features, this application provides an extremely fast and dependable way for non-technical users to create and maintain even the most sophisticated reports. Because the reports are fully data-driven, the user has exact control over the layout and never needs empty "placeholder" columns. e.Spreadsheet Designer supports a virtually unlimited variety of report formats.

With e.Spreadsheet Designer the non-technical user can deliver charting and complex reports with drill-down and interactive capabilities faster than it would take a developer to deliver reports that are much more basic using another approach.

More on e.Spreadsheet Engine and e.Spreadsheet Designer:

e.Spreadsheet Designer tutorial (*basic*):
http://www.birt-exchange.com/modules/wfdownloads/singlefile.php?cid=4&lid=213

e.Spreadsheet Designer tutorial *(advanced, including conditional formatting, charting from data ranges, using dynamic parameters):*
http://www.birt-exchange.com/modules/wfdownloads/singlefile.php?cid=4&lid=94

e.Spreadsheet Engine live demo (*captures input and plugs it into a spreadsheet that resides on the server)*
http://www.birt-exchange.com/modules/wfdownloads/singlefile.php?cid=4&lid=309

Using VBA (Excel Macros) with e.Spreadsheet Designer:
http://www.birt-exchange.com/modules/wfdownloads/singlefile.php?cid=4&lid=178

Sheet bursting with e.Spreadsheet:
http://www.birt-exchange.com/modules/wfdownloads/singlefile.php?cid=4&lid=140

e.Spreadsheet Data Range Mechanics (*a tutorial focused on data manipulation):*
http://www.birt-exchange.com/modules/wfdownloads/singlefile.php?cid=4&lid=144

To download a free 90-day trial of e.Spreadsheet Engine or e.Spreadsheet Designer:
http://www.birt-exchange.com/modules/wfdownloads/staticdls_index.php

More Information

Email: essinfo@actuate.com

1-800-884-8665

www.actuate.com