Aspose.Total for .NET

Vibrant Software

Custom Software Integration

Adam Miller, October 26, 2009

Product Background / Overview

Vibrant Software is a leading technology solutions company in Southern California. We help our clients reduce costs and create workflow efficiencies through custom software and software integration services. We develop open and scalable systems and as a Microsoft Partner, we have a strong technology expertise in the Microsoft realm. Most of our clients utilize Microsoft Office as their primary business desktop software and expect the reporting capabilities of any custom software package to work seamlessly with their existing Office applications. We needed to find a 3rd party toolset that let us easily build custom reports that worked natively with the existing Microsoft Office infrastructure found in our corporate client's offices.

Requirements Scenario

We needed to take existing Word documents, Excel spreadsheets and PowerPoint presentations and render them dynamically from data contained in a database. In many cases, the reports were very consistent reports with matching data fields or data elements from the databases that we were designing. Our requirements were separated into two categories:

Business Requirements:

1) Output reports must support Office 2003 and Office 2007
2) Specifically, the output must support Word, Excel and PowerPoint (and Visio - TBD)
3) Licensing model must be per developer (similar to Visual Studio or MSDN)
4) No runtime licensing fees for our clients

Developer Requirements:

1) Simple to use the toolset – our existing developers must be able to learn it quickly and become productive quickly
2) Proven product history – we wanted a package that's proven and stable
3) Good API for interfacing with the documents
4) Fast and scalable – We could have thousands of users using the environments we build, so it has to be very responsive

Solution Implementation and Benefits

Approach

Vibrant Software had two clients that both needed the extensive reporting capabilities of Aspose at the same time. We were able to leverage our learning to accommodate both clients as well as create a common reporting framework and a common set of helper code classes to facilitate easy reporting needs.

We approached our solution design by first analyzing the specific intended reports for each client. We determined that the reports fell into 5 major categories:

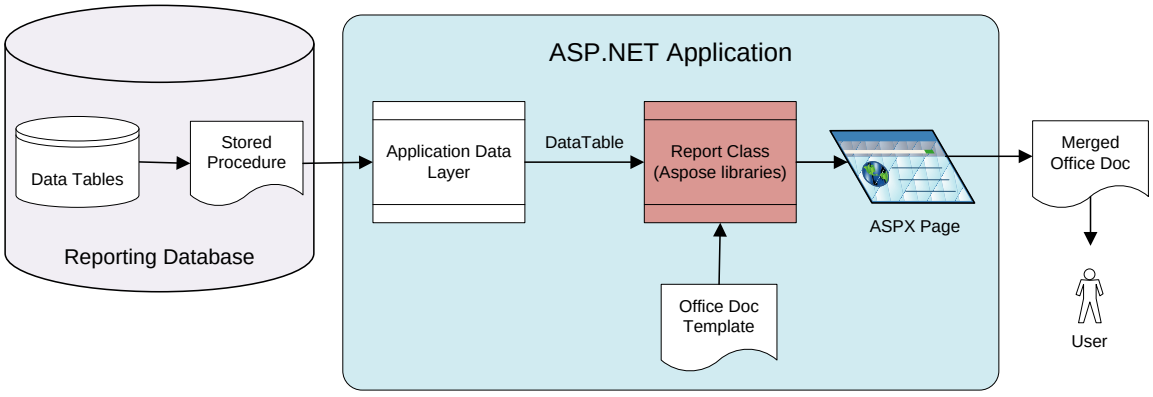
1) Word Documents
2) Excel Documents
3) PowerPoint Documents
4) Visio Documents
5) "Other" Documents

The Word, Excel and PowerPoint documents were the easiest to plan for. They're natively accommodated with Aspose and there are many examples. The Visio documents were more challenging, though. At this time, Visio is not supported as an output format, so we migrated all of our Visio documents over to PowerPoint. In many cases, we could actually copy/paste from within Visio and paste directly into PowerPoint.

The final category of "other" documents included all of the documents that were not easily turned into templates. This included one-off reports or reports with a large amount of user added content that was not stored in our databases. We chose to leave these documents outside of the solution at this time.

Implementation

We chose to implement a common pattern for all of our reports. Each report followed this architecture:



This architecture gave us many advantages and allowed us to create a scalable environment that was easy to learn and maintain for future developers. Many of the advantages include:

1) Functionally separated code: The Aspose libraries were able to be separated

into a C# library that was compiled as a separate project. This allowed us to "hide" the implementation of the Aspose libraries away from the rest of the code. In the event we ever needed to change libraries (not that I see that happening, but it's a good practice to follow), it would be much easier.

2) The data that was used for the reports was retrieved through a common set of SQL Stored Procedures. We were able to associate one main Stored Procedure with each specific Report (Word, Excel or PowerPoint) and it was easy to understand the flow of data.

3) The Office templates were named a specific naming convention and we then named our Report Class with a similar name. This allowed us to easily associate a given report with the business logic (report class) and with the data source (Stored Procedure).

Rendering

One of the benefits of a dynamic reporting system built on the Aspose components is the ability to render the reports in their native format (Office Word, Excel or PowerPoint) OR the ability to save to PDF. This allows us to build a single reporting engine for users – even if they don't have the final Office products on their workstation. In our UI, we have a dropdown that lets the user select either the Native format, or convert to PDF.

Office Formats

One of the problems we foresaw was the need to stay current with the versions of Microsoft Office and the document formats. Aspose manages that for us, and we don't have to worry about changing formats by Microsoft or functionality changes. For some of the more sophisticated Excel reports, we are able to publish data directly to a tab, and then have pivot tables and other lookup tables utilize that data. This gives us very dynamic graphs and pivot tables in native Excel formats, but still driven from the database data via Aspose.

Consistency

One of the challenges we foresaw was the ability to implement various report types (Excel, Word, Excel) with a common toolset. The Aspose utilities really create a consistent programming experience by allowing our developers to implement the various report types with very similar code. We found a few scenarios where the consistency could be improved (creating PDF files for example), but overall the API for all modules was well written and organized.

Future Implementations

Our clients are starting to use the applications now and are very excited about the reports. The irony is that the more reports we create, the more they want. I foresee the majority of the future development centering around enhancements to the

existing reports and adding new reports to their reporting library. In the next release of the Excel components, better support for Pivot tables is possibly going to be introduced and we have some reports lined up to take advantage of those features.

Conclusion

Overall, our experience with the Aspose products has been very positive. The products have met all of our expectations and our requirements outlined in the beginning. The API consistency really allows us to quickly build different report types and allows us to bring new developers into the project pretty quickly. The pricing model is reasonable and the rendering speed is very good. Overall, I see the Aspose products remaining as a solid component of our toolset.

Excel Chart Example Screen Shots

The following screenshots represent an example of an embedded chart in Excel and how we provided the data to the spreadsheet dynamically. We chose to allow Excel to manage the graph and the rendering of the graph. We provided the data on a separate tab that was then used as a data source for the graph itself (**Figure 1**). We used the CalculateFormulas API Aspose method to update the graph after putting the data into the spreadsheet tab so it would render instantly (**Figure 2**).

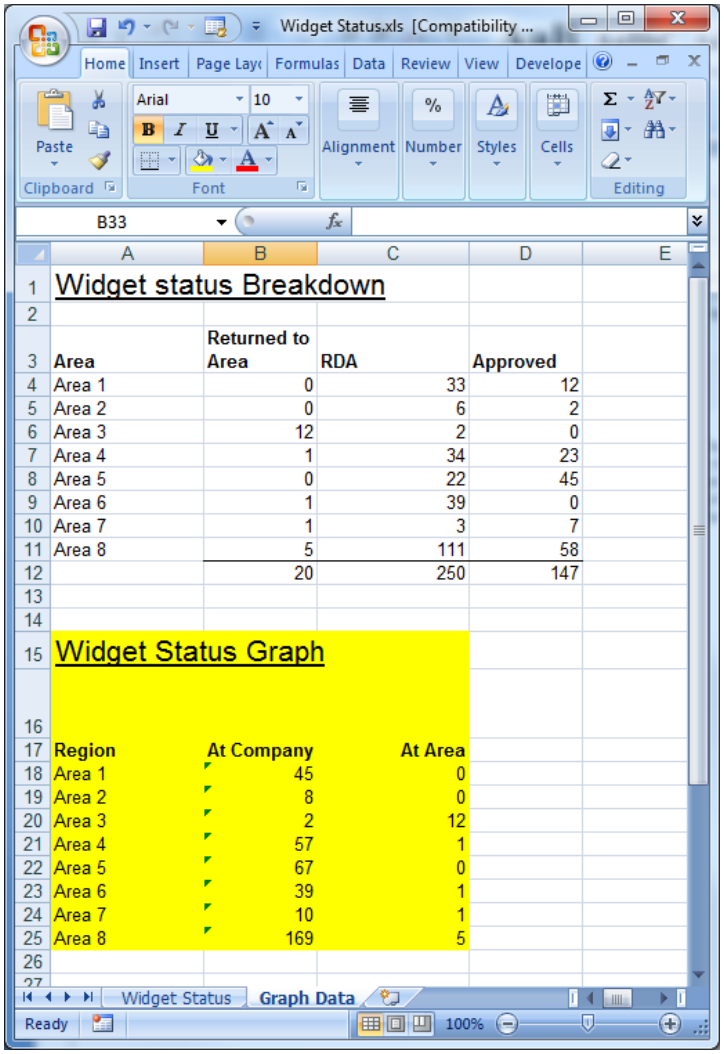


Figure 1

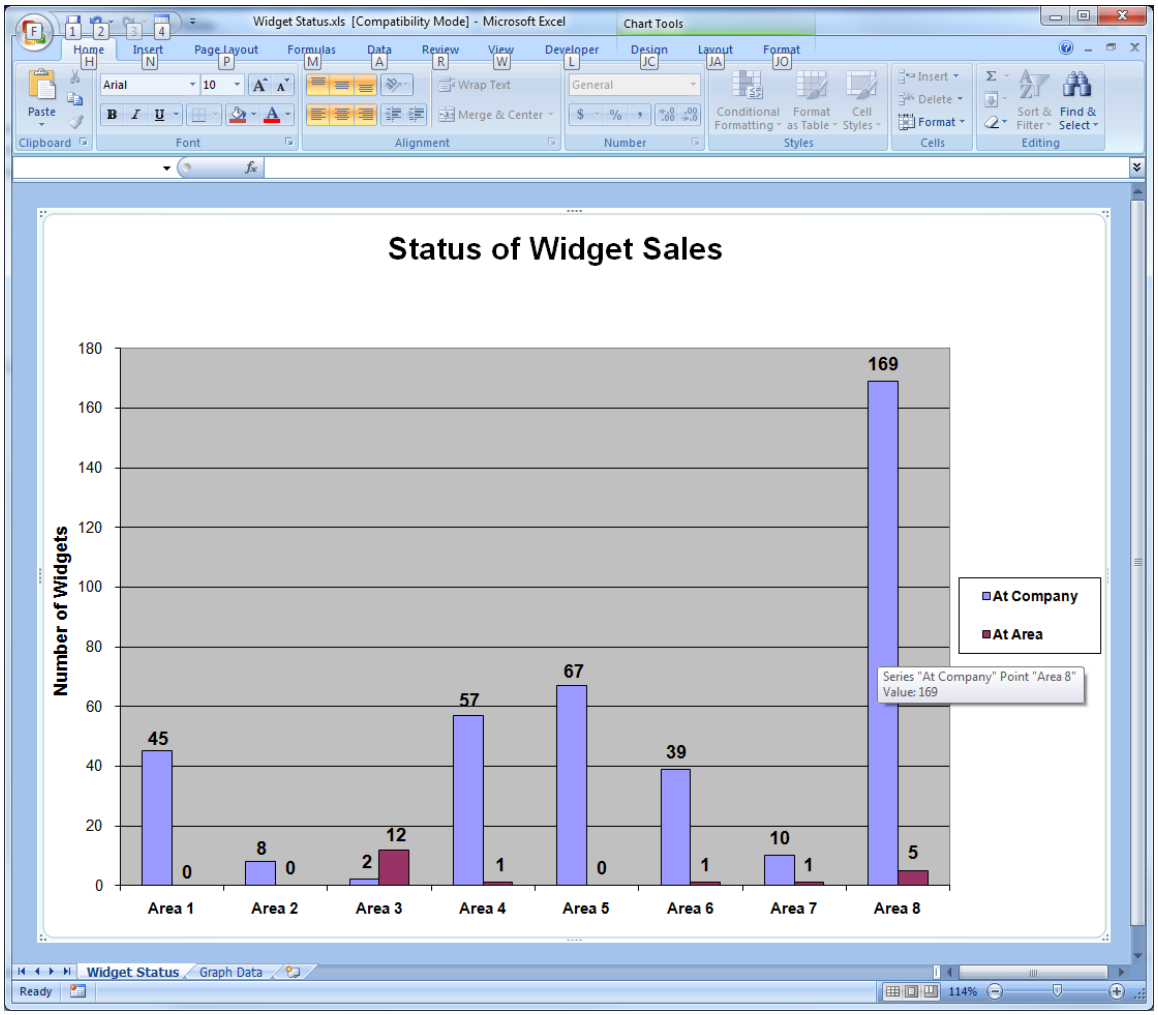


Figure 2

Visio to PowerPoint Example Screen Shot

The following screenshot represents an example of a report that started in Visio, but was converted to PowerPoint for us to render dynamically. All of the labels and the counts are provided from a database and placed into the diagram via the Aspose libraries at runtime. This report was copied and pasted from Visio to enable it to run in PowerPoint.