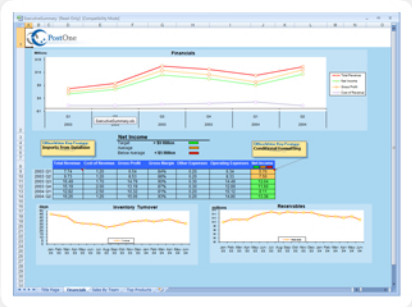
SoftArtisans OfficeWriter:

Enhanced Integration for Microsoft Business Intelligence



WRITTEN BY:
ANDREW J. BRUST

BLUE BADGE INSIGHTS

softartisans OfficeWriter

Contents

Introduction

SoftArtisans' OfficeWriter takes Microsoft's Excel and Word and transforms them from mere spreadsheet and word processing applications into full-fledged data presentation and Business Intelligence delivery platforms.

OfficeWriter dates back to the early days of Microsoft's Active Server Pages technology. I remember when the first version came out, as I was new to Web development and I wanted a way to return data in Excel format without resorting to generating simple CSV (comma separated values)-formatted files. SoftArtisans' ExcelWriter offered just the solution I needed, and I was duly impressed.

More than a decade later, the ExcelWriter product endures, though it's changed quite a bit. It now, of course, is a .NET-based product and, along with WordWriter, has become a component of OfficeWriter, which provides support for both Word documents and Excel workbooks.

But the growth is more profound than that. OfficeWriter now integrates with SQL Server Reporting Services (SSRS) and SharePoint. It offers comprehensive programmable object models for Excel and Word, and offers an add-in to each of these applications to make the SSRS integration even better. OfficeWriter is now a full-fledged Business Intelligence (BI) product, even as it remains a highly sophisticated developer tool.

Over the years, I've become very specialized in BI myself. And as I also work as an analyst focusing on the Microsoft stack, I'm keenly interested in products that enhance the stack's value, especially when it comes to BI. It is with all that in mind that SoftArtisans and I provide this review of OfficeWriter, which covers not just the product's features and capabilities, but also the context of their importance to Microsoft BI.

Custom Office Solutions, And Why They Matter

Microsoft Office is tightly embedded into the world's businesses. Office file formats are standards in and of themselves: most budgets, financial models and even status reports are produced in Excel, and virtually all letters and contracts are produced in Word. Delivering reports as data-driven documents in Office format lets business users feel ownership of the data and a greater intimacy in their understanding of it. It garners more attention and creates more buy-in.

The Office applications themselves are familiar to people, so people feel comfortable exploring information within the apps, instead of just consuming it. Providing a report in HTML or PDF format is at best like sitting at an intimate table in a large restaurant. Providing a report in Office format is akin to serving a meal in someone else's kitchen, and letting them help cook. OfficeWriter provides those home-cooked reports in Office format, and does so in both business user- and developer-friendly ways. As this review progresses, I hope you'll appreciate how nicely this works and how valuable it is.

Reporting on Data, Actionably

Excel is arguably the world's most pervasive analysis and calculation tool. While not always the ideal tool for *storing* data, it is an excellent tool for data discovery. Features in all recent versions of Excel,

like Charts and PivotTables, as well as Excel 2007 features like Data Bars and Conditional Formatting, and Excel 2010 features like Sparklines, together make Excel a very powerful analysis tool.

Word is not used for data presentation as frequently as is Excel, and its data visualization and analysis features are not as rich as those of its spreadsheet cousin. However, for documents that are truly documents – i.e. those which require precise pagination, margins, text flow, and other word processing niceties – Word is a great reporting vehicle.

So Excel and Word file formats, then, are effective delivery containers for data-driven content. But while they do have provision for the import of external data (for example, using the Data tab on Excel's ribbon or the mail merge facility in Word), integrating data deep into the fabric of a word processing document or a spreadsheet is difficult to do through such import facilities. Going further can require venturing into more code-oriented approaches.

VBA and VSTO: OK on the Client, No-Go on the Server

Import facilities are a start, but a more precise method of data manipulation is accomplished through the use of macros written in Visual Basic for Applications (VBA) or, for more experienced developers, .NET code created with Visual Studio Tools for Office (VSTO). Solutions developed using these two technologies take advantage of a programming object model that provides full access to workbooks or documents, their formatting and their content, allowing data to be interspersed in the way the developer or customer requires.

But VBA and VSTO are not the perfect tools for producing documents and spreadsheets as reports. There are a number of reasons for this. First, writing code in VBA, VB .NET or C# (the last two of which are the languages VSTO code can be written in most easily), is not especially easy for the business users and information workers who need to prepare these reports. Second, VBA and VSTO are ill-suited to many scenarios where documents *need* to be produced though code, because many such scenarios involve code running on the server.

In Web applications, or those where a high volume of documents need to be created, documents are best produced in server-side processes. But VSTO and VBA require the actual Office applications, making them unsuitable for server execution. For reasons of stability, scalability and speed (as well as licensing restrictions on using client applications in server processes), generating the file content directly, rather than manipulating Office applications to produce them, is what is required. VSTO and VBA, therefore, do not supply the right solution.

How can we generate Office files without using Office itself? Microsoft offers a seemingly plausible solution: an API (Application Programming Interface) for its newer file formats, called the Open XML API. But while this API is appropriate for server-side execution, its object model's paradigm is much more oriented to the Open XML schema, file layout and objects than to concepts at the higher level of Word documents and Excel workbooks. It also does not work for the Office 97-2003 binary file formats.

Where does that put us? We have now considered Office import facilities, VBA, VSTO and the OpenXML API, but none of these options provides the full range of features, file format-compatibility and server-side performance that we need. Yes, Office gives us choices, but it also forces difficult trade-offs.

OfficeWriter to the Rescue

This leaves a big gap, and OfficeWriter from SoftArtisans fills it. It does so by providing an application-level object model, compatibility with the older binary and newer Open XML file formats, and accessibility to business users, all on a server scale. OfficeWriter provides a user-accessible data templating framework *and* a programmable object model, both of which are server-friendly and run completely independently of the Office apps themselves.

The common need for server-generated, Office-formatted reports, OfficeWriter's ability to produce them, and the unique ways in which it does, is compelling on its own. But OfficeWriter's capabilities have important facets and applications beyond what we've already described. As previously stated, OfficeWriter integrates with SQL Server Reporting Services, and this allows it to further integrate with a number of data sources including SQL Server Analysis Services and even non-Microsoft data sources like Hyperion Essbase, SAP BW (a.k.a. NetWeaver BI), Teradata, and Oracle. Further, OfficeWriter's integration with SharePoint allows it to publish SharePoint list and Business Connectivity Services data in Office documents, and push all of these data-driven documents and workbooks out via SharePoint to business users, for maximum collaboration with minimal code.

In this review, we will examine these complimentary capabilities of the product. We will examine OfficeWriter's use as a developer tool, as a BI tool with Excel and as a data-driven document and presentation tool with Word. We'll also look at how OfficeWriter significantly enhances the value customers can derive from existing, and often significant, investments in Microsoft technology.

The OfficeWriter Developer Story

Business Intelligence software, in an industry-general sense, tends to seclude itself as a specialty, niche category of technology. Many BI tools are expensive, stand-alone and, even if they are products from the so-called "mega-vendor" software companies like IBM, Oracle and SAP, they exhibit traits of their acquired pure-play BI company heritage. Similarly, most mainstream software developers also view BI technology as specialized, and thus separate from, and irrelevant to, their application development and relational database work. This is a pity, because BI is most powerful when integrated with mainstream technology, in mainstream applications, with implementation participation from mainstream developers.

OfficeWriter is one of a select few tools on the market that bridges the mainstream-niche BI gap, because it is a mainstream developer tool and a BI tool. We'll cover each of these two sides of the product, and we'll begin now with the developer category. As you read through this section and become familiar with the product's programmability, keep in mind that one of its ultimate destinations (and one of yours, by using it) is the delivery of BI content, in a form easily accessible by, and actionable to, business users running your application.

As we have already mentioned, OfficeWriter's underlying object model is exposed both in a programmable form as well as through a templating framework. We'll take a look at each interface, and we'll also discuss how to use them together, but let's first consider why the availability of both is sensible and advantageous.

Imperative vs. Declarative: Two Views of Data-Driven Documents

Throughout the last 20 years or so of programming and in the world of word processing as well, there have been two approaches to getting things done. One involves the issuance of imperative commands; the other involves a more declarative and often visual method of specifying what is needed. This contrast is demonstrated nicely in Web page design: you can edit a Web page's HTML markup directly and then see the impact by viewing the page, or you can edit the page through a visual designer, and accomplish the editing and viewing simultaneously.

The visual approach is commonly referred to as WYSIWYG, which stands for "what you see is what you get." A computer science professor of mine once referred to the command-based, imperative approach as YAFIYGI, which stands for "you asked for it, you got it." OfficeWriter offers you similar choices. On the imperative/"YAFIYGI" side lies direct programming against the OfficeWriter object models. With this option, you write imperative code to create or modify a document's content, formatting and/or structure. On the declarative/"WYSIWYG" side lies the templating framework, wherein you insert special placeholder codes (mail merge codes in Word, or special Data Markers in Excel) that will later be substituted with specific columns/fields from data sources that you bind to the document. The YAFIYGI approach is geared to developers who prefer to integrate application data into a document through explicit commands that govern the data's placement. The WYSIWYG approach instead allows OfficeWriter to take an existing document and insert specific data items at specific positions within it.

The *ExcelTemplate* Object: for Power Users and Developers

Figure 1 shows a simple Excel spreadsheet prepared for use with the *ExcelTemplate* object. It consists of a title graphic and text, and a simple 4-column table, with column headers and a row of content, followed by a single blank row. The main row of content contains four data markers, each of which is positioned in one of the table's columns. The place holders reference a data source called *ProductQuery* and, from the leftmost column to the rightmost column, the *ProductNumber*, *Name*, *Subcategory* and *ListPrice* fields within that data source.

When this document is rendered, the data markers will be substituted with the actual data values for their corresponding data fields. What's more, if the data source contains more than one row of data, a spreadsheet row will be inserted after row 6 (the row containing the data markers) for each additional row in the data set. **Figure 2** shows the rendered output.

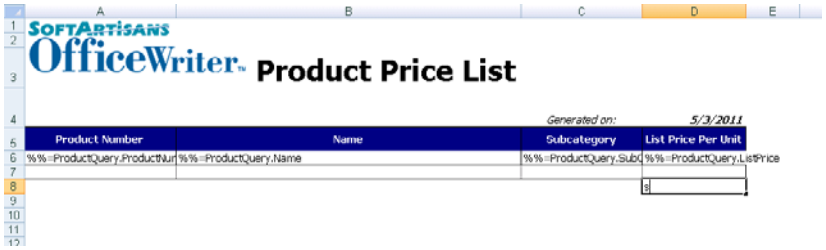


Figure 1

SoftArtisans OfficeWriter Product Price List

Generated on: 5/12/2011

Product Number	Name	Subcategory	List Price Per Unit
CA-1098	AWC Logo Cap	Caps	$8.99
GL-F110-L	Full-Finger Gloves, L	Gloves	$37.99
GL-F110-M	Full-Finger Gloves, M	Gloves	$37.99
GL-F110-S	Full-Finger Gloves, S	Gloves	$37.99
GL-H102-L	Half-Finger Gloves, L	Gloves	$24.49
GL-H102-M	Half-Finger Gloves, M	Gloves	$24.49
GL-H102-S	Half-Finger Gloves, S	Gloves	$24.49
LJ-0192-L	Long-Sleeve Logo Jersey, L	Jerseys	$49.99
LJ-0192-M	Long-Sleeve Logo Jersey, M	Jerseys	$49.99
LJ-0192-S	Long-Sleeve Logo Jersey, S	Jerseys	$49.99
LJ-0192-X	Long-Sleeve Logo Jersey, XL	Jerseys	$49.99
SB-M891-L	Men's Bib-Shorts, L	Bib-Shorts	$89.99
SB-M891-M	Men's Bib-Shorts, M	Bib-Shorts	$89.99
SB-M891-S	Men's Bib-Shorts, S	Bib-Shorts	$89.99
SH-M897-L	Men's Sports Shorts, L	Shorts	$59.99
SH-M897-M	Men's Sports Shorts, M	Shorts	$59.99
SH-M897-S	Men's Sports Shorts, S	Shorts	$59.99
SH-M897-X	Men's Sports Shorts, XL	Shorts	$59.99
SH-W890-L	Women's Mountain Shorts, L	Shorts	$69.99
SH-W890-M	Women's Mountain Shorts, M	Shorts	$69.99
SH-W890-S	Women's Mountain Shorts, S	Shorts	$69.99
SJ-0194-L	Short-Sleeve Classic Jersey, L	Jerseys	$53.99
SJ-0194-M	Short-Sleeve Classic Jersey, M	Jerseys	$53.99
SJ-0194-S	Short-Sleeve Classic Jersey, S	Jerseys	$53.99
SJ-0194-X	Short-Sleeve Classic Jersey, XL	Jerseys	$53.99
SO-B909-L	Mountain Bike Socks, L	Socks	$9.50
SO-B909-M	Mountain Bike Socks, M	Socks	$9.50
SO-R809-L	Racing Socks, L	Socks	$8.99
SO-R809-M	Racing Socks, M	Socks	$8.99
TG-W091-L	Women's Tights, L	Tights	$74.99
TG-W091-M	Women's Tights, M	Tights	$74.99
TG-W091-S	Women's Tights, S	Tights	$74.99
VE-C304-L	Classic Vest, L	Vests	$63.50
VE-C304-M	Classic Vest, M	Vests	$63.50
VE-C304-S	Classic Vest, S	Vests	$63.50

Figure 2

The Business User's Declarative Environment

The amount of code that would be required to generate the output shown in Figure 2 without the template is not huge, but for a business user, it wouldn't be insignificant either. Most business users likely wouldn't write such code, but could easily produce the spreadsheet shown in Figure 1. Simply by placing data markers in the format %%=*rowset.columnname* within a sheet, business users can easily map out what data should be placed where. As long as all data markers for a given rowset appear in the same spreadsheet row, the report is ready to have data bound to it and render the final report.

Binding Data to Workbooks

There are a couple of ways to go about supplying data to the spreadsheet. One of them does require writing code, but very little. The core C# code required to bind the data, then render the populated spreadsheet as *PriceListRendered.xlsx* and return it to a Web user to be opened in Excel is shown in **Listing 1**.

```
ExcelTemplate xlt = new ExcelTemplate();
xlt.Open(Page.MapPath("templates/PriceList.xlsx");
DataBindingProperties basicBindingProperties = xlt.CreateDataBindingProperties();
xlt.BindData(PriceListDs, "ProductQuery", basicBindingProperties);
```

```
xlt.Process();
xlt.Save(Response,"PriceListRendered.xlsx", false);
```

Listing 1

The code in Listing 1 assumes that a) the *SoftArtisans.OfficeWriter.ExcelWriter* assembly has been referenced, b) its namespace has been imported and c) an ADO.NET *DataSet* called *PriceListDs*, containing the price list data, has already been created and that its column names are identical to the field names referenced in the spreadsheet's data markers.

BindData can be called multiple times, with different data sources passed in its first parameter each time. For each *BindData* call, the same *DataBindingProperties* object can be used or unique ones can be created, each ostensibly with different property value settings. The *DataSource* passed to *BindData* can be an ADO.NET *DataTable*, *DataSet,* an object array or any object that implements *IDataReader*.[1]

Grouping and Nesting

Simply populating a rowset of data into a sheet, as shown in Figures 1 and 2, is fine, but sometimes data is more hierarchical than such a structure supports. For such scenarios, the Enterprise Edition of ExcelWriter supports grouped/nested data. Markers like %%group, %%header and %%footer are used to group and format data; ExcelWriter looks for a standard data marker in the same column as the %%group data marker to determine which column to group on.

There's more to the *ExcelTemplate* object than we can cover in this review, but the overall concept should now be clear: Excel can be used as a powerful report writer against a wide variety of data. Most of the work is done in the spreadsheet templates themselves, requiring only the few lines of code shown in Listing 1 to integrate the reports into an application.

WordTemplate: Bringing Structured Data to Documents

SoftArtisans' WordWriter product, and its *WordTemplate* object, does for word processing documents what *ExcelTemplate* does for spreadsheets: it allows business users to author standard documents, and insert special markers to indicate how external data should be integrated into the document at a later time. The *WordTemplate* object looks for standard Word mail merge codes (rather than the data markers used by *ExcelTemplate),* and the Enterprise Edition handles grouping through the use of standard Word bookmarks, which define bindable, nest-able "repeat blocks."

Figure 3 shows a page from a Word document template containing seven mail merge codes, which are contained within a single bookmark, called "Catalog." The "Catalog" bookmark contains additional bookmarks within it to allow for the hierarchical relationship between Category, Subcategory and Product to be handled properly in the document. **Figure 4** shows a page from an output document rendered from the template, and **Listing 2** shows the code required to generate the output. Notice the square bookmark brackets surrounding the various merge codes in Figure 3.

---

[1] The *ExcelTemplate* object also has a *BindRowData* method that can be passed ADO.NET *DataTable*, *DataSet* or *DataView* objects, object arrays or .NET objects which implement *IDictionary* or *IEnumerable*. The last three of these can also be passed to the *ExcelTemplate* object's *BindColumnData* method.

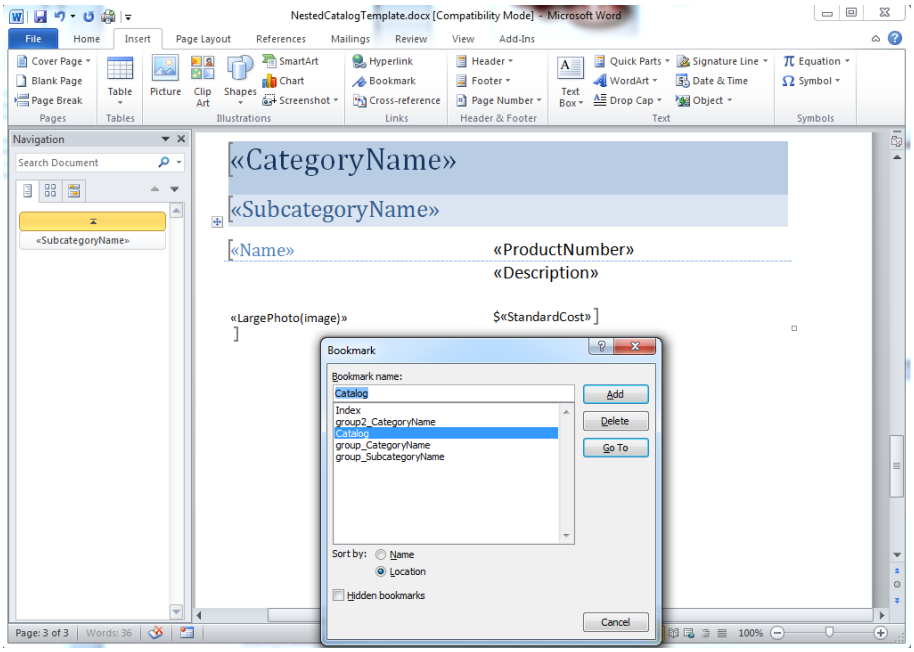


Figure 3

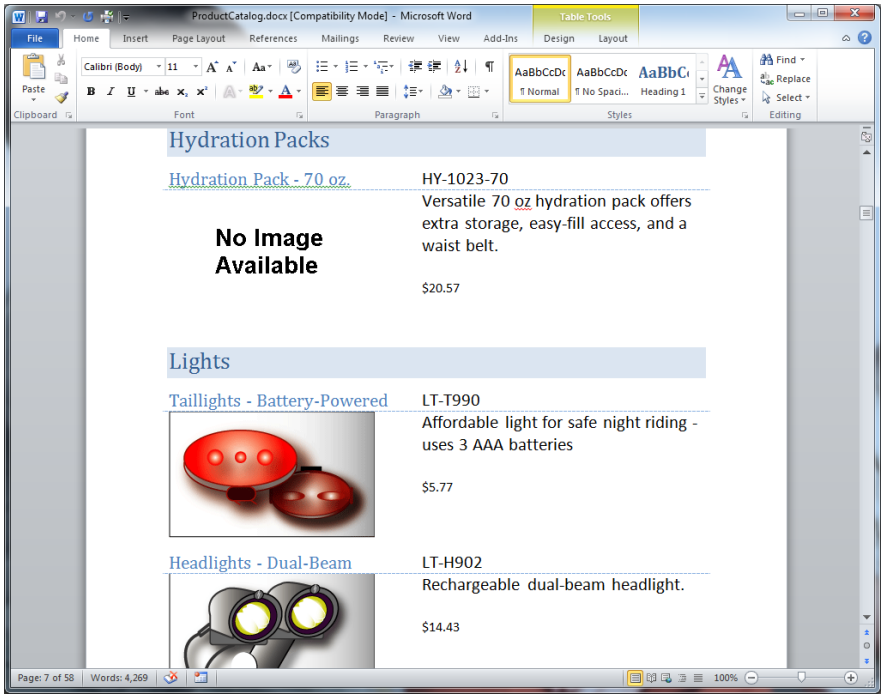


Figure 4

```
WordTemplate wt = new WordTemplate();
wt.Open(MapPath("templates/NestedCatalogTemplate.docx"));
wt.SetRepeatBlock(GetCatalogData(), "Catalog");
wt.Process();
wt.Save(Page.Response, "ProductCatalog.docx", false);
```

Listing 2

The code in Listing 2 assumes that the *SoftArtisans.OfficeWriter.WordWriter* assembly has been referenced and that the function *GetCatalogData* returns an ADO.NET *DataTable* object containing data corresponding to that function name. The call to the *WordTemplate* object's *SetRepeatBlock* method binds the *DataTable* object to the "Catalog" bookmark shown, among others, in the Bookmark dialog in Figure 3.

The *WordTemplate* object's *SetRepeatBlock* method, as it's used in Listing 2, is analogous to the *ExcelTemplate* object's *BindData* method as it's used in Listing 1. The *WordTemplate* and *ExcelTemplate* objects have the methods *Open*, *Process* and *Save* in common.

OfficeWriter's *ExcelTemplate* and *WordTemplate* objects provide enormous power. They can be integrated into an application with very little coding, and later in this review we'll see how Reporting Services integration can bring their power to bear without any code at all. Meanwhile, if you *want* to write some code, the Template objects provide impressive programmability, allowing for simple implementation or tight application integration.

Coding All the Way: Using the OfficeWriter Application Objects

But what if WYSIWYG isn't your bag? What if it's "YAFIYGI or bust" for you? Then the *ExcelApplication* and *WordApplication* objects are what you seek. These tools allow you to create a workbook or word processing document in code, with all content and formatting programmatically determined. You can insert data directly, or you can even insert data markers (or merge codes in the case of Word documents) programmatically and then use the Template objects to populate the data. As a developer, you have full control.

Figure 5 shows a partial page of a data-driven document built completely from code using the *WordApplication* object. It contains data from Microsoft's AdventureWorks sample database; specifically, it shows orders by salesperson.

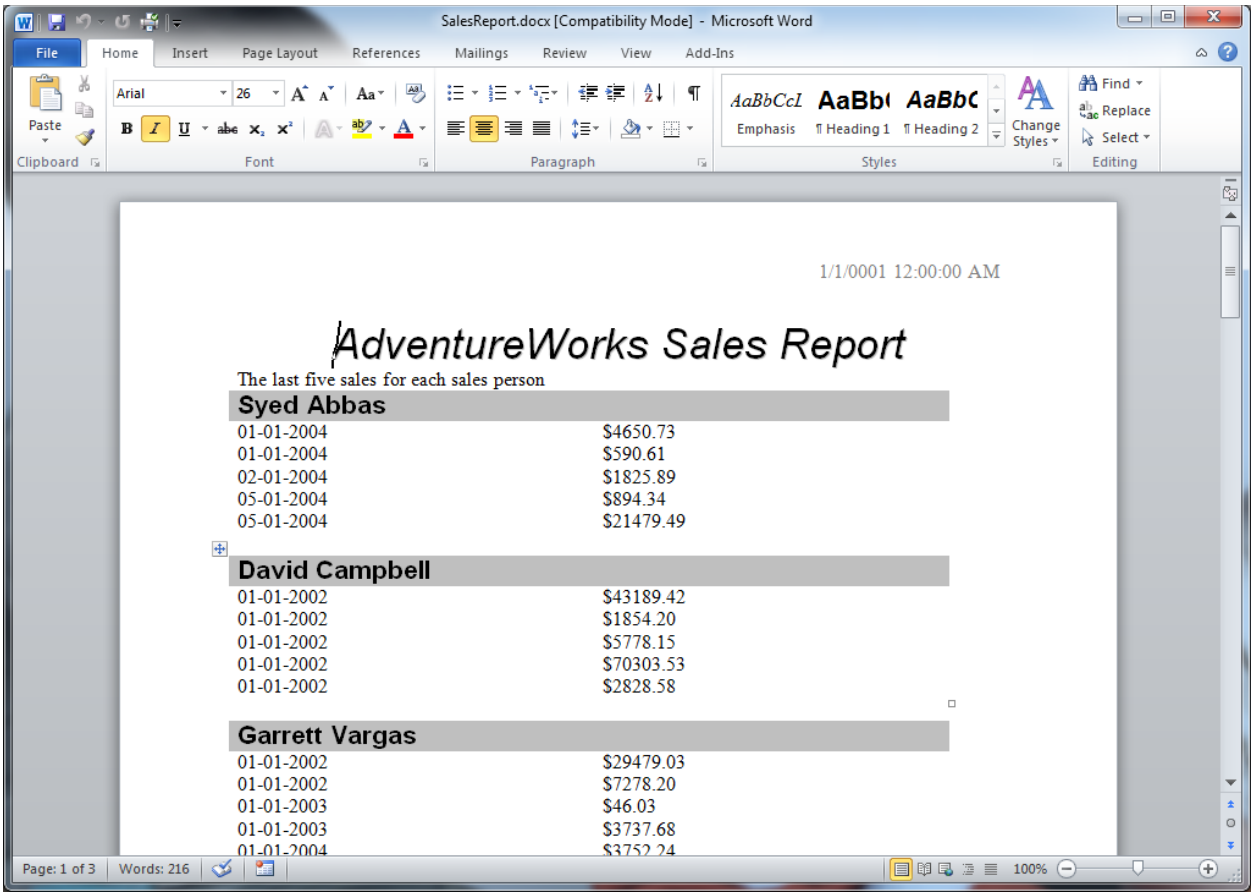


Figure 5

The code used to produce this document uses WordWriter *WordApplication*, *Document*, *Table*, *TableCell* and *Shading* objects to create a formatted table that will be populated with data. Within the table, the code loops through a *DataReader* with salesperson data to create the section for each salesperson, inserting the salesperson's name. Within each salesperson's section, the code inserts merge codes for date and sales amount and creates a bookmark containing them both. The code then loops through all the bookmarks it created and, using a *WordTemplate* object attached to the *Document* object in memory, binds a *DataTable* with sales data for the corresponding salesperson, using the *SetRepeatBlock* method.

Figure 6 shows a manually created Excel workbook, waiting to accept data. **Figure 7** shows a populated version built using ExcelWriter's *ExcelApplication* object.

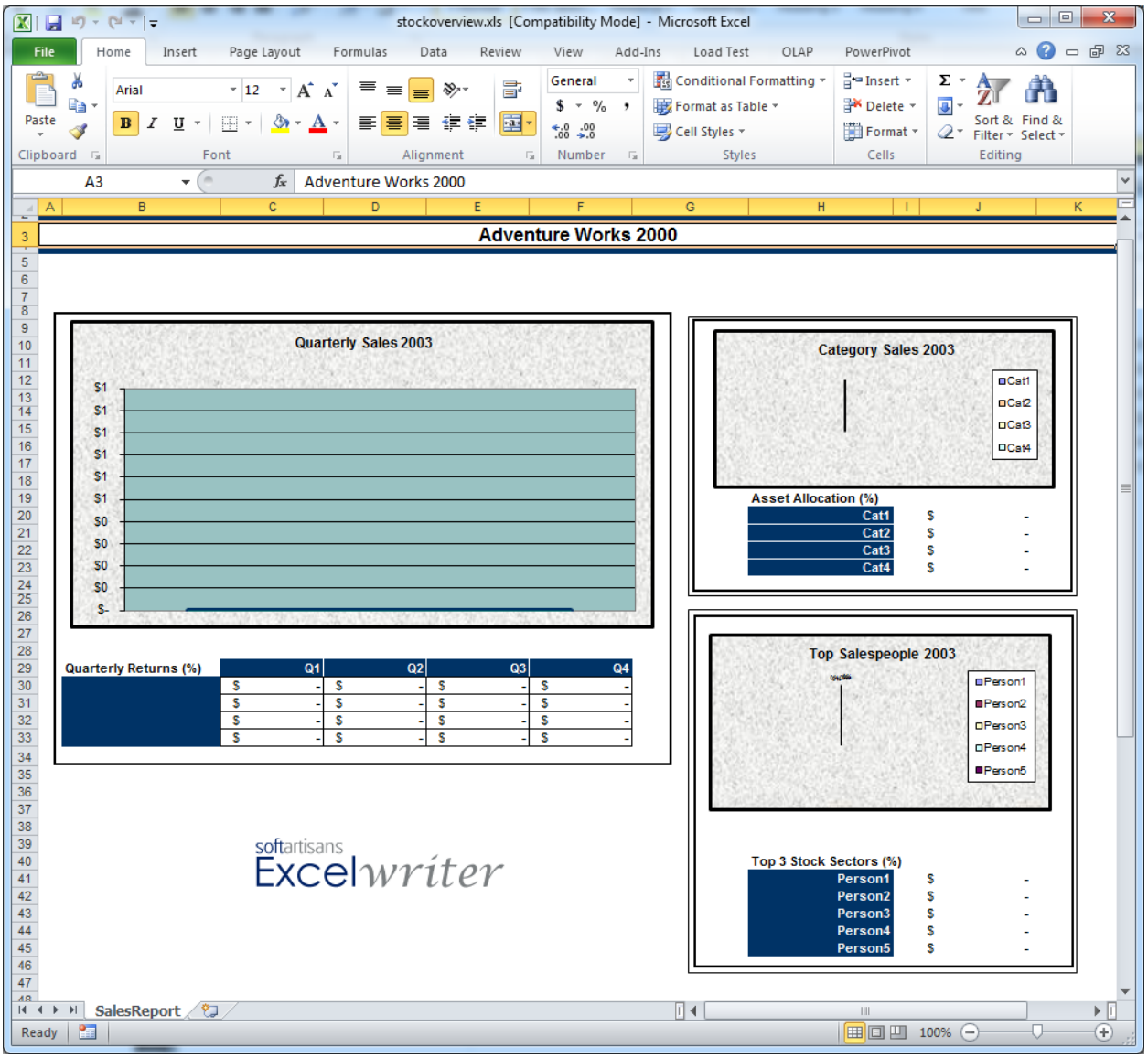

stockoverview.xls [Compatibility Mode] - Microsoft Excel
Adventure Works 2000
Quarterly Sales 2003
Quarterly Returns (%)
Q1
Q2
Q3
Q4
Category Sales 2003
Cat1
Cat2
Cat3
Cat4
Asset Allocation (%)
Top Salespeople 2003
Person1
Person2
Person3
Person4
Person5
Top 3 Stock Sectors (%)
softartisans
Excelwriter
SalesReport
Ready
100%

Figure 6

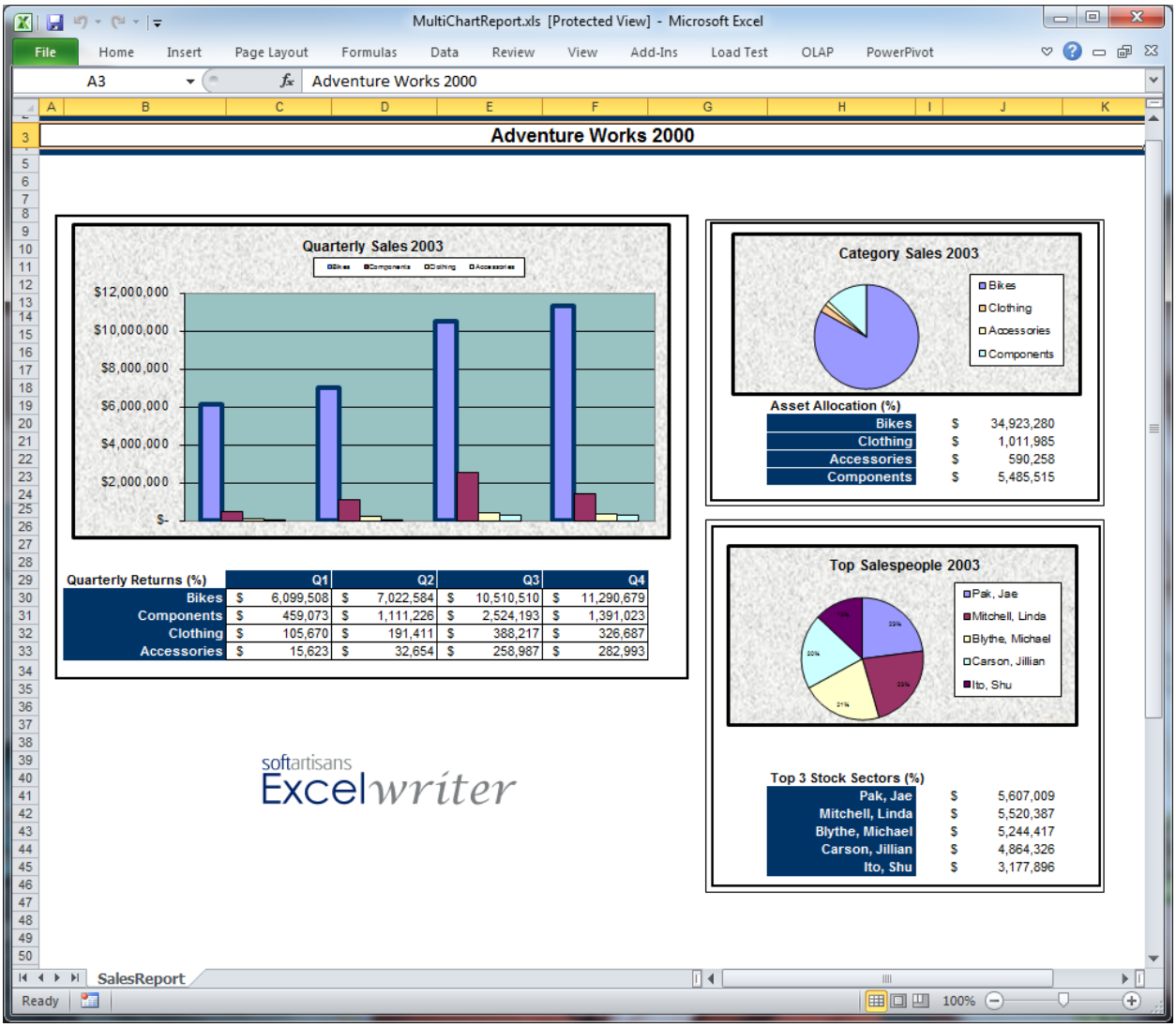


Figure 7

ExcelWriter objects including *ExcelApplication*, *Workbook*, *Worksheet* and *Chart* were used, along with ADO.NET *DataTable* and *DataReader* objects, to build the content shown in Figure 7.

Although the initial spreadsheet in Figure 6 was created manually and updated by the *ExcelApplication* object, workbooks can in fact be created completely from code, completely in memory and streamed back to the user without a physical file ever being saved to disk on the server. So 100% YAFIYGI solutions are possible, and for relatively text-heavy workbooks, they are quite feasible. However, for highly graphical workbooks, where attractive output is required, manual layout can make for better aesthetics, as in this case.

Developers are Data Specialists

What we've seen in this section of our review is that OfficeWriter is extremely developer-friendly, allowing completely code-driven solutions, low-code power-user solutions, and hybrids of the two. Were these features the full extent of OfficeWriter's capabilities, it would already be an extremely

useful tool. But the product goes well beyond that; its integration with SQL Server Reporting Services, and use of Excel and Word as both authoring and delivery vehicles, make OfficeWriter a standalone Business Intelligence tool. Used in this mode, no code is required whatsoever. We explore this aspect of the OfficeWriter product next.

The OfficeWriter BI Story

BI professionals have a love/hate relationship with Microsoft Office, and especially with Excel. BI pros appreciate the power of Excel and the degree of autonomy it provides users, thus relieving central IT from the burden of producing highly customized analyses whose importance may be high, but whose usefulness requirements may be temporary. Excel is a powerful query tool; its charting engine provides excellent data visualization functionality, and features like conditional formatting and Sparklines allow users to turn spreadsheets into bona fide scorecards.

Excel can also function as a quasi-database, in that the individual sheets in a workbook can store data and act comparably to tables in a relational database. For some users, this functionality provides temptation to use Excel as if it were genuinely a database, creating full-fledged BI solutions that are completely self-contained within workbooks. By, in effect, creating a data mart in a spreadsheet – sometimes humorously referred to as a "spreadmart" – business users unwittingly abuse Excel by taking it into scenarios beyond its intended design.

In the love/hate relationship, the spreadmart phenomenon catalyzes the latter. There are several reasons for this, each a manifestation of where Excel falls short of a true database. To begin with, data in spreadsheets will become "stale" (and thus inaccurate) unless frequently updated. Data in spreadsheets is also subject to erroneous modification, whether accidental or intentional. Worse yet, spreadsheets are often shared as email attachments, resulting in multiple copies of spreadmarts delivered to individual users' hard drives, each of which is subject to the vulnerabilities just described.

Things get really bad when large volumes of data are stored in workbooks…because while Excel can query them, it does not possess a true database engine. In fact, the performance Excel delivers when querying data in its own workbooks can be particularly deficient.

We've seen that OfficeWriter can help. We've seen how it safeguards against Excel's downsides while allowing business users to take advantage of Excel's analysis and data visualization capabilities. We've seen how OfficeWriter populates Excel workbooks with data stored *externally*. Whether through the use of data markers and a little code using the *ExcelTemplate* object, or a purely imperative coding approach using the *ExcelApplication* object, OfficeWriter delivers on Excel's promise as a data presentation vehicle, while keeping the source data itself out of the spreadsheet.

What we haven't seen yet is how OfficeWriter can be used in conjunction with SQL Server Reporting Services (SSRS), such that the query, orchestration of rendering, and management/delivery of subscriptions are handled by the SSRS infrastructure. Query design is also "offloaded" to SSRS, specifically the Visual Studio-based report designer or the standalone Report Builder tool. Because the data sources bound to spreadsheets are the queries within the reports, no data binding code is required.

And because both Report Builder and the Visual Studio report designer have their own user-friendly query designers, no SQL skills are required either. What's more, SSRS can connect to Analysis Services cubes and its query designers can generate the Analysis Services MDX (MultiDimensional eXpressions) needed to query them. That means that even OLAP (online analytical processing) reporting is enabled by OfficeWriter, making it a true BI tool and not merely an alternative relational report writer.

The OfficeWriter-SSRS integration consists of three OfficeWriter capabilities:

1. Add-ins for Word and Excel provide the ability to open existing reports, and select queries from them (in other words, select from one of the existing datasets already defined in the report)
2. Using the same data markers supported by the *ExcelTemplate* object and the same mail merge code and bookmark conventions supported by the *WordTemplate* object, users can indicate where within a document they would like specific fields from the selected query(ies) to appear. Users simply utilize data markers and merge codes to point to the query they selected and columns within that query. With the Enterprise Edition, and using an add-in button, SSRS formulas can also be inserted in Excel workbooks and Word documents. For workbooks, Excel formulas can be inserted as well. Business users and others may find the latter to be more powerful and easy to use.
3. Reports can be run interactively through the add-in. But they can also be run from any SSRS report viewer environment at run-time by exporting a report and selecting "Excel designed by OfficeWriter" or "Word designed by OfficeWriter" as the output format. Report subscriptions that are configured with those output formats can also be used for report delivery.

OfficeWriter makes all this possible by storing the template Excel workbook (i.e. the one with the data markers) or Word document (i.e. the one with the merge codes), as embedded files within the Reporting Services RDL file. This means that any combination of native SSRS, WordWriter and ExcelWriter content may be contained together within a single RDL file, and that the SSRS engine can produce any or all of the renderings on-demand or on a subscription basis.

These capabilities accommodate a variety of scenarios including:

- Using the SSRS designers exclusively for query design, then subsequently designing OfficeWriter reports in Word and/or Excel – effectively delivering a no-code version of the *WordTemplate* and *ExcelTemplate* objects' experience.
- Taking pre-existing, richly designed SSRS reports and re-purposing their queries to build analyses or reports in Excel or Word, quickly and easily, with the OfficeWriter SSRS add-ins.
- Building rich, native SSRS reports and using OfficeWriter to build faithful likenesses of them in Word and/or Excel. The OfficeWriter "ports" provide a way to render the original report's content in Word/Excel without the loss of quality, detail or interactivity that often occurs when such rich reports are exported to Word/Excel using SSRS' native export capabilities.

No matter which of the above scenarios you utilize, OfficeWriter lets you use Word and Excel as BI presentation technologies without the downside of storing data in spreadsheets or limiting yourself to simple mail merge scenarios in Word. Your production databases can be used as they would be from a

conventional SSRS report, but you gain access to the data visualization and analysis features of Excel, the layout and pagination features of Word and the added relevancy to users of delivering reports as Office documents. You will not be creating spreadmarts or even storing your reports in separate Office files, yet the *output* produced is in native Office file formats that users appreciate and identify with.

A Split Personality?

When you think about it, it's really uncanny how a single product can be used in such widely varying capacities. The fact that OfficeWriter is an extremely powerful developer tool on the one hand, and yet functions as a completely stand-alone BI tool gives it a versatility that very few other products achieve.

If we take that thinking a step further, we see that this versatility is less about mutually exclusive use cases and more about an opportunity to mix and match them. We can imagine Reporting Services reports containing (1) Word and Excel templates that function nicely as application-embedded reports and (2) native RDL content that allows the report to work in a standard Reporting Services deployment, be it in SSRS standalone or SharePoint-integrated mode. We can even imagine OfficeWriter reports that are initially implemented as *ExcelApplication* or *WordApplication*-generated, then get ported to SSRS-based templated reports and then have a native SSRS report implementation added afterwards.

OfficeWriter's mission is to accommodate the entire YAFIYGI/WYSIWYG spectrum, and to deliver Office-based output in application-integrated and enterprise reporting environments. Because whether you're embedding reports in applications, or allowing end-user authoring and execution, using Excel and Word as the output format is just plain smart. So why not have a common toolset for both kinds of reporting?

Not Just Docs: OfficeWriter Turns Word into Pitchbook Engine

Excel is such a data-oriented tool, and OfficeWriter for Excel works so nicely with it, that the value proposition of OfficeWriter for Word may seem only incremental, at first. Sure, purchase orders, product catalogs and the like make their case for using the product, but the usefulness of OfficeWriter for Word goes beyond those applications.

In a number of businesses today, and especially in financial services, pitchbooks – page-or-screen oriented presentations that sell a product or an idea – serve as very important tools for analysts, business development personnel and marketing specialists. But pitchbooks aren't easy.

The thing that makes pitchbooks so useful and, often, so effective, is that they meld together highly improvised content and layout on the one hand, and intersperse up-to-date, relevant data, on the other. This is also what makes pitchbooks hard. The need to make pitchbooks data-driven often requires some amount of code and automation. But the potentially inconsistent, irregular and ever-changing format of the documents makes the YAFIYGI code-based approach to injecting the data difficult to apply. If a marketing person can't tell a developer exactly where specific data should go, or can't have the discipline to keep to the spec given to the developer in that regard, then how can the developer write the code to update the document? The pitchbook scenario makes for code that can be difficult to write,

and very brittle once it is written. Clearly then, a WYSIWYG approach is necessary, and OfficeWriter delivers.

The marketing folks designing pitchbooks and the developers keeping their data up-to-date need not fight. Using OfficeWriter for Word, the document designer can build the document any way she'd like. As long as she inserts merge codes and bookmarks into the positions where she wants the data, and as long as she shares the merge code and bookmark names with her developers, all is well, and everybody wins. Developers need not worry about document layout – they just need to map data sources to merge codes and bookmarks. Even if those merge codes and bookmarks get moved around in the document, the code still works. You could say that OfficeWriter enables "loose coupling" of data and documents, and that's exactly what pitchbooks require.

This loose coupling brings a flexibility whose importance should not be underestimated, and the importance of it is particularly applicable to the use of Office documents as output formats. Almost since the dawn of PCs, there has been a natural tension between business users and developers. Business users need things to be flexible – this stems from the agility they must exhibit to work effectively, competitively and to win new business. Developers need things to be navigable, consistent and repeatable – this stems from their need to work efficiently and avoid otherwise needless re-work, especially since that effort is better spent on brand new applications. For the purposes of maintainability, developers need requirements to be as close to static as possible; but for the purposes of value, business users' needs must be flexible and dynamic.

What's great about OfficeWriter is that it lets business users maintain the dynamic canvas of Excel and Word to get what they need done. But instead of posing a moving target to developers, OfficeWriter bridges these documents to the more predictable, repeatable world of programming. This removes the business user-developer tension, and lets everyone get their jobs done. This versatility is great for pitchbooks, as well as a larger variety of data-driven documents.

Maximizing Investments

What we have seen throughout this review is that, when it comes to BI, many different Microsoft products and technologies provide important functionality and power. To start with, of course, SQL Server is a strong, industry-leading database and its Analysis Service component makes affordable BI available to a very large installed base. But the strength of the stack goes well beyond the data repositories: Reporting Services is an excellent query and delivery engine for reporting, both against relational and OLAP data sources; SharePoint works well for simple data lists and the sharing of reports and documents; and Office applications, especially Word and Excel, are so universally used and understood that using them as a publishing channel can put BI in everyone's hands.

The problem is that, out of the box, these products and technologies do not work as cooperatively as they probably should. Yes, Reporting Services can be used to report on data in SQL Server and Analysis Services, but non-programmer business users may have difficulty designing these reports. Moreover, publishing them in their native format, or exporting them to PDF, or even static Office formats, limits

their utility to information workers. Reporting Services integrates with SharePoint, and so does Office, but ironically these products cohabitate in SharePoint more than they collaborate there. Office itself has some important data connectivity capabilities, but mail merge in Word and the ability to bring in query result sets, in bulk, into Excel can only help information workers use their data to a limited degree.

While these products have integration points between them, they don't reach the full potential for harmonious integration that they could. Each product provides significant value, to be sure. But there's untapped latent value too, especially in the way products might work together. The Microsoft BI stack whole is somewhat better than the sum of its parts, but it could be better still. A lot better.

That's where OfficeWriter comes in: it immediately adds value and utility to Office, but then it goes one important step further. OfficeWriter takes a collection of Microsoft products, including SQL Server, SharePoint and Office, and creates a positive feedback loop between them. This empowers businesses to fulfill the real BI vision: taking raw corporate data, and enabling all participants in the business – not just specialized analysts – to understand it and make insightful decisions based upon it. And because some or all of these products are widely deployed in small, medium and enterprise businesses around the world, almost any company can deploy OfficeWriter and increase the value of its Microsoft stack investment significantly.

This is what good third party products should do: build upon widely-deployed big products from major vendors, and add a level of refinement that extends the product's functionality and greatly enhances the product's value. So called "mega-vendors" work best at implementing major functionality; their partners tend to execute best by understanding the application of these technologies in the market and augmenting their functionality in ways they know customers will appreciate. Mega-vendors need their partners, and vice-versa. The cooperation underlies a symbiotic relationship between the two parties that mirrors those in nature. That's why the partner environment is called an "ecosystem." OfficeWriter is a prime example of why it's so important.