Part I: Introducing Excel Services

1

Introduction to Excel Services

So why do you care about Excel Services? Well, since you bought this book, there is a chance your organization uses Excel in its day-to-day operation. Who can blame them? Excel is the most popular modeling tool and the most popular database tool. It is so versatile that the same person can use it for both a complex financial model and a simple task list. In the Excel organization, for example, it is not uncommon to use Excel for managing tasks, project milestones, and bug reports. Over the years, Excel has gathered a very large set of features ranging from advanced formatting to advanced data acquisition mechanisms. With Excel 2007, this set of features has been bolstered even more to allow Excel to be a first-class BI (business intelligence) tool.

Why Use Excel Services

While Excel is a great tool, it really lacks in one specific area: it is a client application. It was designed to be a client application from the get-go, and in all probability it will stay that way. That means it is focused on one user getting whatever Excel functionality that user needs on a PC. There are many indicators showing that organizations will want to run Excel on the server:

- For one, a lot of people go to great lengths to try to get Excel client to work in a server environment — at great cost and with great frustration.
- People want one version of the truth — but when workbook files are used with Excel client, there is no real protection against people modifying them. With only the client at the users' disposal, it is much harder to keep a single version that will be the "single point of entry" to the data.
- Intellectual property is expensive, and companies want to guard it. Excel models can become extremely complex and give a real edge to their owners. Those owners do not want others to be able to access the models — only the results of the models. Excel does not really supply such protection.

- Running a lot of models, whether as part of a mechanized process or because a lot of users need to get at the data, is virtually impossible to do in a scalable manner with Excel.
- To see any part of an Excel file, the entire file must be opened. This can put a strain on even the fastest networks and can be completely impractical when people are connecting over a WAN.

More accurately, in some cases Excel is smart enough to delay-load some types of data caches such as pivot tables.

- People want to see and navigate Excel worksheets inside a browser. But they also want those worksheets to be up to date, and they want the ability to navigate them and do simple operations such as drilling down through information or filtering lists.
- Administrators want more manageability of what Excel does. Some workbooks have complex data queries that, when executed by too many users at the same time, can bring databases to their knees. Conversely, some data sources are accessible only to specific users. For these reasons, workbooks that are distributed may sometimes only have copies of data rather than actual live data. This raises the "one version of the truth" problem — how can organizations know what data is current?

For these reasons and others, organizations end up producing various creative solutions to the problem. These solutions usually come in two flavors: large, custom-built farms of Excel client applications and rewriting the underlying models.

The Excel Client Farms

Excel farms are relatively large farms (or computing clusters) that run multiple instances of Excel on request. People usually build some kind of protocol that allows them to extract information from the Excel processes and transfer it back to the user in whatever form is desired.

These solutions, while creative and impressive, do not scale well and are not fun to maintain. They require a lot of hardware due to various Excel limitations (again that pesky "was not designed for the server" thing), and managing the whole thing is really hard to do.

Rewriting the Models

The other solution organizations come up with, which is no less painful (and in some ways, more painful), is to take tried-and-true models written in Excel and rewrite them in some other language so that they can be used without Excel. This is of course a huge time investment — not only can the models be excessively complex, but when a model changes, the code needs to be updated appropriately, which takes a lot of time (testing alone is a major time sink, since one needs to make sure that the model behaves the same way that it does in Excel).

Note that, in some cases at least, the reason for transferring models into code does not have to do with the inability to get server capabilities but rather is done to squeeze every ounce of performance out of a model. Excel Services may or may not help in these cases. Some organizations will retain their need to rewrite Excel models.

How Excel Services Comes to the Rescue

Excel Services has been created to solve all these problems and more. While still using portions of Excel code that have been rewritten to be serverworthy, large amounts of work has been done around that code to make it into an actual server product.

Excel Services solves some of the problems simply by virtue of being a server product — intellectual property is protected because users do not have access to the actual workbook (unless the admin allows them to see it). Since all the calculations are done on the server, it is not even an issue — there is no need to transfer the model itself to the user. This also takes care of the "one version of the truth" issue — there is a central repository for information, and only people who are allowed to update that repository will. Furthermore, because Excel Services is leveraging the SharePoint infrastructure, it can make use of such features as "view-only rights" where some users can access the complete file (by loading it in Excel or by saving it to their hard drive), while others can only view it through the server.

Because it is a server, it can also do various things that the client was never designed to do, such as sharing information across users. The file itself need not be loaded more than once. This not only reduces network traffic, but it also means that the actual process of loading the file, which can be very long and CPU-intensive for large models, is only done once. This is doubly true for models that rely on getting data from databases — Excel Services can figure out what data is sharable among which users and make sure that it does not query the database too many times. On top of that, the administrators can also instruct Excel Services to only hit the database server periodically — say, not more than once every 5 minutes — giving them much more control over how many times the database will be hit by requests.

Since Excel Services was built from the ground up to have multiple instances of the same workbook open at any given time, it is possible for multiple users to open any number of workbooks and work with them. The same goes for processes that need concurrent access to workbooks. The number of workbooks that can be interacted with at a given moment is only limited by memory and CPU.

Finally, because the server supplies the means to access parts of loaded workbooks, people who are across the WAN will not need to take the hit of loading the entire file — they can just request a small part of the workbook. This also ties into EWA (Excel Web Access), which is a Web Part that allows people to navigate Excel workbooks inside a browser (*no* ActiveXs, I repeat, no ActiveXs at all, just plain old HTML and JavaScript, honest).

Excel Services Goals

It is important to understand what our goals were when we started thinking and designing Excel Services. Understanding the goals explains a lot of the technical decisions we made throughout the project. I am only listing the goals that have specific bearings on our server product — obviously goals such as customer satisfaction are there by default.

First Goal — 100% Fidelity with Excel

Our first and foremost goal was to have 100% fidelity with Excel in every Excel feature we support. That is to say, if we support a given Excel feature, our goal is to support it in exactly the same manner that Excel 2007 supports it. It does not mean, though, that we support all of Excel's features.

That said, sometimes when things are transferred to a server environment, "100% fidelity" stops being a clear-cut thing. As an example, take the Data Refresh feature. In Excel, when the user refreshes the data, Excel will go and grab new data from the database back end. On the server, as described before, the administrator can place a limit on how often refreshes may occur.

Second Goal — Security

It is incredibly important that Excel Services does not expose information to unauthorized users. If a user does not have access to a workbook, that person should never be able to use Excel Services to circumvent that principle (unless the administrator specifically set up the server for that). Similarly, if a workbook is set up such that users should not be able to see the actual models and formulas of the workbook, Excel Services should never supply that information to users.

Additionally, users should never be able to see information that belong to other users. So, if a user accesses data and gets a set of results and a second user gets a different set of results, neither of them should be able to see the results of the other.

Third Goal — Robustness and Reliability

Since this is a server product, robustness and reliability are paramount — we want to be able to keep the server up and running for as long as possible. If there is a feature in Excel that is impossible to translate well to the server because it will reduce robustness or reliability, there is a good chance we will preemptively decide not to do it at all.