BACK TO THE FUTURE, OR FORWARD?
© Never2Busy.Com

At the very beginning of on line computing, those who dared interact with the computer were faced with green screen terminals that displayed little boxes to type into, function keys to control processing, and a blinking box called a cursor that indicated where on the screen the computer was currently focused. Only data elements predefined by the programmer were accepted for processing, and only the queries of data previously defined by the programmer were available for viewing. Report content was hard coded and printed in fixed proportion fonts in rigidly spaced lines in black on paper with alternating green and white bars. There was no way to rapidly share information with others and it was just too bad if a 14" printout didn't fit in a #10 envelope. You could pay the postman or lug it yourself. Data exchange between applications was possible only through custom "interface" files.

Then came personal computing. The lure and promise of the personal computer has been "my data, my way". The famous 1984 SuperBowl ad by Apple Computer depicting a hammer throwing woman breaking a giant screen and interrupting a rigidly controlled presentation clearly captured the attitudes and imaginations of an entire generation of personal computer users.

The entry of IBM into the marketplace legitimized the concept and allowed for the rapid penetration of computing into the workforce. Fueled by programs like VisiCalc, business users enjoyed unprecedented ability to manipulate, connect and display data according to their dictates. Microsoft caught the vision and re-built its MS DOS operating system into the Windows operating system; introduced thousands of new features to encourage the "my data, my way" movement; and led the shift away from predefined processing sequences executed in a batch mode, to processing sequences determined solely by "events", the order and nature of which where entirely controlled by user actions such as clicking a mouse, or connecting a cable.

But as the complexity of personal computer hardware and software increased, so did the complexity of the code required to manage the diverse operating environment created by users. The response of the software industry to this problem was a gradual movement of the user's application to a "host computer" where the operating environment was more easily controlled. Users were then given access to their data through a "client" program, which contained only the minimum necessary elements to access the "approved" data elements from the "host computer".

Initially, "hosts" were located within corporate data centers and connected by private cabling, but the expansion and bandwidth of the internet has enabled a whole new model, called "cloud computing" where the user and their application/data are connected over the internet, and thus, can be located anywhere in the world

As the processing of data was being moved to "host" computers, the "client" side programs became increasingly structured through the use of on screen "controls". (The term controls refers to the broad class of on screen methods of interaction between users and their application, and includes things like data entry fields, drop down screens, check boxes, etc.) Furthermore, these "client" programs began to become "thinner" and more standardized. Thinner has been touted as better, because thinner clients require a smaller resource footprint in the user computer, allowing them to run on a broader range of computer configurations. The advantage for corporate IT is that the thin client can run on a wider range of computers than the historically "fat" client, allowing mass deployment of cheaper

machines and a longer life cycle for older machines, as well as allowing for a broader range of supported devices –laptops, internet appliances, and cell phones.

For example, a complex client application might require a minimum of 2 GHZ, 64 bit processor with 1 GB of RAM, a sound card and viewing screen with a predefined resolution, along with 100 GB of hard drive space to contain it. A corresponding thin client might only require a 500 MHZ, 32 bit processor with 256 MB of RAM, and no hard drive—if it was entirely contained on the host computer and downloaded in pieces only when needed.

Standardization of clients has simplified the job of planning, installing and supporting applications for IT departments. Users have also benefited because they only have to learn how to operate a single program.

In searching for the perfect "standardized" client to support a client/host model, many IT organizations have embraced Internet Explorer. Its advantages are obvious:

1. It comes pre-installed with the Windows OS and is available on a wide range of platforms- no extra cost, no extra install time and it gets along well with the other toys in the sandbox.

2. It supports extensive customization and can update itself automatically for new features.

3. It represents an industry standard.

4. It can work with any host data source.

But is this the right thing for users?" Does this tool advance the "my data, my way" philosophy that launched the industry, or does it move us backwards to a prior era?

Personally, I have to say that my Windows computer screen, running IE as a client, is feeling more and more like that old green screen terminal I abandoned in the '80's when I embraced the "my data, my way" movement.

Sure, I now have a color screen instead of the green, and I have a sound card and graphics support for multimedia. I have a mouse too. That's new. But my keyboard is still the same, and it's still required. I still have my big hard drive, a fast processor and lots of RAM memory, but these resources are hardly used, since all the data manipulation occurs on the host, and when they get used, they are used for system overhead, like swap files, and caches, and not to hold or manipulate my data. At some point, the IT department will come by and declare that all these expensive resources in my desktop, are no longer needed, and will replace my costly PC with a cheaper internet appliance.

And when I look carefully, not only do the on screen "controls" bear a striking resemblance to a bygone era, but I've lost the ability to interact "my way" with "my data". In short—it's like 1970 all over. Nothing significant has changed. It's kind of like being stuck in a cage in the jungle of Vietnam, or locked in a room at Waldorf Astoria—different places, same situation.

Access to my data is being limited to pre-defined functions and views and the flow of data between applications not anticipated by the programmer is again awkward if not impossible.

Is this then, the inevitable end of personal computing?

Not necessarily. I believe there are a number of alternative client programs that retain the "my data, my way" philosophy and still yield efficiency to the IT department. I am not familiar with all of them, so I will reserve my comments to the one that I am familiar with: Microsoft Excel. Most people think of Excel as a nice tool for typing up rows of numerical data, and maybe printing a report or making it into a graph. Excel offers so much more:

Excel, like IE is pre-installed on most corporate PCs, and is supported on a wide range of platforms.

Excel allows customized program modules to be added to its menus and events. Excel, in fact, can be configured to look and act like any Windows program. Excel can read and write to web sites like IE or FTP, connect to host databases and submit queries, produce reports, send E-mail, and even update Windows itself

Excel represents an industry standard.

Excel can read and write any data source. But unlike IE, which---no matter how much javascript or other code is inserted---is still bound by the limitations of the read only hypertext document model, Excel's workbook structure with worksheets containing cells in rows and columns, combined with its natively rich feature content, has inherently greater flexibility to accommodate the real life data structures that users work with.

A client application programmer, using Excel, can give a user unlimited access to their data, but still retain the essential elements and benefits of client-host computing.

Here are a few "my data, my way" actions that users can do natively with Excel that they cannot do with Internet Explorer (unless the programmer has pre-coded the feature!):

1. Re-arrange the data on the screen
2. Add additional content (ad hoc or computed)
3. Merge the content with content from another application or query
4. Make the content available to other applications (by copy/paste, formula link, or file export)
5. Change the headings, titles, paging, footers or format of the data (bold, color, etc)
6. Reformat the data to underscore a point
7. View the data as a chart or graph (or scroll through and preview multiple views of the data in different chart or graph formats, before selecting one)
8. Controls can interact with offstage data –rows, columns, cells, entire worksheets or even other applications!
9. Extend the presentation of the data in new ways by adding sheets, graphs, etc.
10. Be truly portable by persisting an interactive version of the content beyond the life of the source data connection.

For a more comprehensive list of things that Excel can do natively that IE cannot, just open an Excel document and look through the menus!

But the ultimate proof that Excel is really the place to do client computing is this: Consider that the export feature most commonly found on IE clients is to Excel or Adobe Acrobat. Ask any corporate user what their most frequent function is? I know mine is "export to Excel". I was recently in training for Crystal Business Objects 11 and the trainer's response to most format questions was: "*Use the 'export to Excel' function and finish the report in Excel!*" So, why write an IE client when the data is almost always going to be exported to Excel anyway? Let's avoid the effort having to write and process the code twice—once to get it from the server to IE and again to send it to Excel. Why not just put the data in Excel in the first place and let the client have at it!

Does anyone use Excel clients?

While a vocal majority of software suppliers are enthusiastically riding the IE bandwagon, the use of Excel is not unheard of. A number of Kaufman Hall products embrace the concept to varying degrees, having been written exclusively to target the Excel work environment, and Oracle/Hyperion has the SmartView add-in which connects its "hosted' applications data to Excel in ways that are can be pre-defined by the host vendor or the IT department. Lawson Software also provides an add-in which allows users the ability to populate a worksheet with application data. I have not seen whether they provide a pre-built client package or merely the query tool that leaves it up to the client/IT department to develop the Excel based application and features.

But, I was intrigued by the Lawson approach, which serves up data elements by screen code, not by table name. Thus, if you wanted to view a vendor address in Excel, and the vendor master inquiry screen was "AP100" on the host, your Excel query would reference "AP100", and receive the same data back from the query as you would have from an IE based inquiry. The traditional approach is for Excel to connect to a database at the table level, and generate SQL queries directly against the fields. However, this approach requires a more technical background. Serving data by screen, instead, gives the user the data they want, without bothering them with "how" the data was structured and obtained. Clearly, this method makes it easier for non technical people to create client applications in Excel.

If Excel is so good, how did we get to this place anyway?

Essentially, Excel is a victim of its own success. It began its life servicing a niche market, and its domination of that market has created an iron clad association with that market in the minds of consumers. So while it's clear to anyone who has studied the evolution of Excel that Microsoft envisioned a greater role for Excel, the marketplace continues to stereotype the product.

On the other hand, IE, began its life as a tool that clearly mimicked the browse/update model used in mainframe legacy applications. Software vendors easily saw that they could salvage millions of dollars of legacy mainframe software by upgrading to IE, adding some sound, color, or graphics to prove to customers that it was no longer a "legacy" product and sell it in the new Windows economy.

From a development perspective, the limitations of working within the hypertext document model created a structure and discipline that made it easy for software designers and programmers to understand how to interact and present data to users. Because of its incredible flexibility, Excel itself provides no clues as to how it should be implemented. Vendors saw this as a call to invest more expensive technical resources to develop proprietary models for creating interactions with users, and chose the simplest and easiest path—IE. And there were also other issues, such as anti-piracy, and security, that IE lended itself well to resolving.

Users, on the other hand, were becoming increasingly frustrated with the complexity of installing and configuring customized applications which frequently required identifying and resolving application conflicts. They rapidly embraced the simplicity and standardization offered by switching to the preinstalled IE.

But is this good for the user? Just because it happened does it make it right?

What about Excel's limitations?

There is no doubt, Excel has its limitations. Many of these limitations, such as poor data security, lack of forced separation between production and development environments, lack of forced separation between code and data, response time, the VBA development environment—can be minimized or eliminated entirely by a carefully thought out strategy that relies on developer discipline or utilizing a compensating component of the Windows operating environment.

For example, concerns about the limitations of VBA are easily overcome if one utilizes a COM Add-In written in C. Concerns about the lack of clear separation between code and data, or between production and development environments can easily be addressed by a consistently applied policy established at the point of development and carried out in deployment practices.

I have personally used Excel for more than 2 years as a client application with positive experience from both a development perspective as well as end user response. I have not found the perfect theoretical framework yet for developing in Excel, but I am experimenting a lot, and continue to refine my approach while producing client side tools that fully embrace the "my data, my way" philosophy. In the end, I believe this approach will give users tools that are cost effective to code and support; that enhance user productivity; that save companies money, and that are pleasurable to use. Were these not the original reasons that we embraced personal computing, anyway? If so, who would settle for anything less now?