

TO: Bill Gates
FROM: Steven Sinofsky
CC:
DATE: 6/8/93
SUBJECT: Systems Retreat: Chicago Integration

Introduction

Although our applications are designed and implemented in separate business units, it has become critical or us to think of our applications in terms of parts of a whole, where that whole, Office, is greater than the sum of the parts. Chicago offers us an opportunity to redefine what it means for applications to integrate together, and we must fully exploit our advantage to do so. We need to use the momentum created by Chicago to increase our application market share and drive us into new markets. Some difficult trade-offs potentially exist, where an application might need to spend time on efforts that are not viewed as critical to category success, or in re-implementing features that already work adequately in order to gain synergy. These costs are implicit in the strategy, but to be clear this strategy should be priority one.

Integration and synergy will be the paradigm shift that will drive the Chicago wave of applications and the Chicago Office. Fundamental changes in application architecture/factoring or the desktop metaphor will be part of the Cairo-specific releases of the applications.

The remainder of this memo discusses the current competitive landscape and then outlines specific ways in which we can be creative to achieve a new degree of synergy and integration. Although this will sound like the old mantra of shared code, the reality is that by sharing code we will get the synergy we need. In fact, it might even take more resources to develop the components because we will be trying to meet the needs of several applications.

Details on Chicago leverage can be found in two primary sources. The Chicago team has their *Points of Light* document that outlines some specifics that an applications needs to do for Chicago. Chris Graham has a memo that outlines many areas an application can potentially leverage within Chicago.

Competitive Landscape

There are numerous efforts going on in the industry to ship suites of applications: Borland/WordPerfect Office, WordPerfect, Lotus SmartSuite, and other second tier efforts (EDS and SPC bundle, for example). The Borland/WP Office is an interesting arrangement that we do not quite understand, especially given that there is no presentation graphics in the package and the need for the database is secondary. WordPerfect will probably release their own suite that includes Presents. Office, WordPerfect, and QuattroPro, leaving out Paradox. In order for these to succeed beyond the short term, significant work will need to be done to gain the level of integration we will obtain in our applications. It is not clear how much work these two companies are willing, or capable, of doing to gain synergy.

Lotus SmartSuite poses a threat in two distinct ways. First, as a suite of applications SmartSuite 2.0 goes a long way towards a very high-level of integration. OLE 1.0 support (and OLE 2 in the works, though possibly indirectly via a layer or even a competing technology), SmartIcons, live status bar, DocOnline, etc. all show that Lotus is serious about making their applications similar. For the suite purchaser concerned with user-interface the differences between our next version of Office, with Word 6 and Excel 5, and SmartSuite in terms of user-interface similarity will be minimal. Office will have superior OLE 2 integration, and we will need to use that as the competitive advantage, along with traditional category strength. Lotus has promised integration of LotusScript into a future version of SmartSuite. If they are

able to deliver on this, it will put us in the position of comparing two scripting languages, which is far more difficult than a has/doesn't have comparison.

The longer term risk to our applications business is from the second half of the Lotus threat, Notes. The current lack of a short term Notes competitor from Microsoft has put is in a dangerous and awkward position. First, we are spending cycles to forge a pre-Cairo Notes product, which we will have a hard time migrating to Cairo. Second, our applications are being forced to do work to support Notes. A point which Lotus only occasionally fails to mention. Current data suggests that the adoption rate of Notes is not as high as the PR surrounding the product. We must, however, be sure that our short-term product is able to integrate with our desktop applications in a manner comparable to the Notes demonstration scenarios. Work is being done in the Access 2.0 product to help position Access as a Notes competitor. We have since decided to utilize VB as the front end for the product, we should be sure those resources are spent in the most efficient manner. Unfortunately, Access' lack of a custom control technology makes it inadequate as a competitive framework.

Implementation

In order to fully exploit Chicago on the target machine of 4MB, we will need to be exceedingly smart about how we use memory in both our applications and systems. Realistically, the size of OLE 2 means that we will be very tight on any multiple-application integration that involves OLE, and it will probably show very poor (thrashing) performance. This is unfortunate, and I hope everyone will seriously look at making this possible. Historically, our competitors have shown a certain insensitivity to the 4MB machine[1] and we should work to exploit that weakness. Every byte of code that we can share between our applications increases the ability to do interesting integration scenarios on a Chicago machine.

There are limits to sharing and reimplementation, however, I would not go as far as to advise that our applications get rid of SDM.[2] I believe, however, that it is crucial that we share implementations of the standard user-interface elements of our applications and possibly with Chicago. An interesting issue is one of how much we wish to distinguish our applications. For example, should a shared Toolbar implementation be moved into Chicago's public APIs or should it remain in a Microsoft application-specific DLL. If we move implementation to Chicago and the implementations do not meet the needs of our applications, then we should improve the Windows implementation. Since many of these features are being implemented as separate DLLs, it should be possible to leverage development resources in our applications groups to assist in this effort. The next section will outline those features where I think it is crucial that we share both interface and implementation. I also believe there are common features that have not received attention.

I realize that it has been extraordinarily difficult in the past to get groups to agree on a design, let alone share code. IDG has made great strides in getting groups to share design, but in order to get real leverage and 100% consistency it is critical that we share implementation. Chicago and Chicago Office presents us with an opportunity to redefine how we work to obtain such goals. Simply put, the more code we share the more interesting scenarios will be possible on commodity machines. For example, today if Word, Excel, and the FileManager are running there are currently three Toolbar/Status bar implementations. On a Chicago machine, this redundancy pushes a working set that is already stretched to the limit. Also, if we

[1] In fact, the press release and data sheet for SmartSuite go as far as to recommend a permanent swap file on a 4MB machine.

[2] The performance of SDM will always be superior to the Windows dialog manager because of the use of lightweight Windows. Currently, SDM is about 100K so sharing it between Word and Excel is only a slight burden to the system. We should investigate, however, if the improvements made to the Chicago dialog manager make this difference imperceptible to the user. Also, the dialog manager does not offer the same localization and Mac support as SDM. Word 6, for example, uses WLM but continues to use SDM for dialogs. Some applications are still not using WLM.

can share the implementation among our applications, then we can have a more interesting implementation. It is quite possible to imagine that we would like only our applications to share such rich features as the Toolbar, and let the system use a minimal implementation for other ISVs. Chris Graham poses the question as "how interesting an application should the operating system be?"

Integration and Synergy

There are many areas where we have not been aggressive enough at identifying and exploiting in our core applications. In addition, most of the areas that we do work in synergy do not share implementation. Chris Graham has a very complete list of possible leverage points for Chicago applications, which when combined with the list of *Points of Light* from the Chicago team, will make our applications very compelling.

Chris Peters suggested that we develop a grid of all of our applications. We should look how each application can integrate and interoperate with each other application. Of course it is unrealistic to assume that we do a complete n^2 breakdown, I do think that it is important to think of non-traditional integration. These features and demonstrations should have input from marketing and should serve as demonstrations for Chicago Office in the field.

The following are just a few points I wish to highlight that I think are critical for the Chicago product, Chicago applications, and the Chicago Office. I think by accomplishing these we can achieve an integration paradigm shift.

User-interface elements

This is the key area where we need to gain synergy. With Word 6 and Excel 5 we have gone a long way towards nearly identical user-interface due to the work of the Interoperability group and the program management resources that the products dedicated to the issue. The problem is that both groups, and subsequently any group that follows the design without sharing the implementation, implemented these features independently. The different interpretations of the design specification made this inevitable. I think it is crucial that we use Chicago as an opportunity to have a single code base for shared user-interface elements. The reduction in working set from just unifying our Toolbar and status bar implementation could be worth it. Ideally, we could even share the Toolbar bitmaps between applications and have only one copy on disk.

One of the key areas we need to move towards is a single base of OLE 2 user-interface elements, since there are so many and they are so standard. This is something the OLE 2 group finally addressed to some degree, but did not provide an easily shared solution for our applications.

An issue that concerns me is how the Cairo user-interface changes would fit this model. If the Cairo user-interface will require applications to expose more of their internal functionality through a common user-interface, the reliance on a system API becomes more difficult. For example, the property sheet mechanism has the potential to require applications to do calculations and other work in order to display, update, and acknowledge property changes. We need to make sure that changes in the user-interface paradigm do not force applications to implement these user-interface changes exclusively in their own code space. It is not very interesting if the only code we can share is the look of the property dialog.

Although the commodity Chicago machine is a VGA, and Chicago is encouraging developers to take advantage of and be prepared for smaller screens, our applications and tools should think about taking advantage of larger monitors and 24 bit color. In particular, most developer tools will be running on larger screens than VGA so we should be sure to take advantage of this automatically for better window

management, i.e. the display of code, properties, project window, etc. in VB. Applications should also use Chicago as an opportunity to use 386 specific instructions in their 16 bit code, if it would provide a performance benefit.

Shared functionality

Along with sharing the implementation of straight user-interface elements, there are also elements that have significant code behind them that we can also leverage. I make the distinction between a Toolbar user-interface that has code to implement the Toolbar functionality, and functionality that is common across our applications that exposes itself in a user-interface. I do not think there is any reason why our applications cannot share standard implementations of a number of functions. The following are just examples.

file open/save: I believe most applications have signed on to use the Chicago dialogs, which is very good. The functionality gained shows how you can extend the semantics of user-interface much more easily with a single implementation, in this case the addition of links and long file names.

insert picture: Currently each application implements Insert Picture with a common user-interface, but there is substantial code that it takes to implement the function. In addition, if we shared the implementation we could do clever things like take advantage of the DocFile layout our applications use to locate additional pictures. We could also utilize this implementation in the shared OLE server ClipArt Gallery.

routing slip: This is another example of user-interface with substantial semantics behind it. Assuming a single implementation, it would be much easier to extend this functionality in more interesting routing scenarios without requiring each application to do significant work.

auto-save/backup: This is an area where our applications and customers cannot seem to agree. Since many customers view this feature as critical, we should develop a super autosave/backup feature and share it among all of our applications. The use of OLE 2 automation to drive the saving and backup means that our applications will need to agree on top level IDispatch interfaces for saving and loading, among others.

VB for Applications (Shell Programmability)

It goes without saying that VBA, Visual Basic for Applications, will be one of the key application, and system, features that we will use to distinguish Microsoft products. The challenge here will be to provide reasonable interfaces (IDispatch programmable ones) that show a coherent strategy. As we saw with Excel 5, and to some extent OLE 2, it is possible to factor things too much and abuse the class/method paradigm. I do not think we will have this coherent view of the world if all of the groups that are currently working on describing interfaces to their product continue on the same path. It takes evolution and experience (iteration) to get these factoring correct. I think it is crucial that we funnel all product interfaces through a single group for review. This review should not try to design the interfaces, but should concentrate on general principles such as naming, factoring, and other conventions.

In addition to programming our applications, it is crucial that we introduce a notion of shell programmability. The Chicago Shell should expose a set of IDispatch interfaces to allow all operations that are allowed from the user-interface. This can be as straight forward as having IShell with methods to accomplish everything needed. It is very easy to turn this into a major design exercise trying to factor folders, files, links, etc. into classes, with collection/container semantics. This would not accomplish what nearly all ISVs need and would only increase the complexity of what is essentially a *batch* file. I think that many of the most interesting scenarios for the Chicago Office will involve the programmability of the Shell.

Currently the Chicago Shell is not a true OLE client site. There would be serious performance problems with both speed and working set if the Shell were to be implemented as a client. Chicago has done some work in demonstrating that with a few additional APIs, mostly in the area of being able to know about a file without reading in all of the contents, it might be possible to use OLE. Currently, Chicago has implemented an OLE-like interface for Shell extensions. Ideally, this would be a true OLE 2 container. This would enable more direct OLE 2 drag and drop support as well as the ability to use the Shell as an in-place activation container. In order for this to happen, the OLE team needs to do some performance work and implement these additional APIs. I think this is more important than OLE adding new features, except for possibly OLE controls. IBM has done some nice things for customizing the Workplace Shell, and we should be aware of that competition.

Within the Shell, we need to have a better understanding of how both file viewers, for browsing in the Shell, and view files, for sending in mail enclosures fit in. The Chicago specification is vague on where viewers fit in the current plan. It is important to define this architecture and give the applications an opportunity to write their own viewers. Microsoft needs a view file architecture to compete with third party technology that is on the way (Adobe Acrobat, Common Ground, Disk Paper). The mail group should probably define the architecture, though the applications are in the best position to define a self-contained view file. I think Microsoft applications should define a special stream of compound files that would contain the code for view files, and the Shell should understand this in the ShellExecute API.

One area that the Chicago specification addresses is the notion of *personal directories*, which are directories that a user can specify for storing documents. Applications should definitely make sure this feature will satisfy their users, and they should then take advantage of it.

Cue cards and Wizards

Unfortunately today there is no standard way to implement either cue cards or Wizards. Each group has their own implementation, and the implementation is usually ad-hoc and extended as each new Wizard is developed. Excel and Word have indicated that their plans are to implement Wizards and add-ins for the Chicago products using VBA. This is exactly what we need to have happen. Given that direction, it becomes possible to implement Wizards that span application boundaries. This would be a major advantage for Microsoft.

Cue cards are a more complex problem since there is more content authoring than automation. Many think that cue cards are a better learning tool for users, because they do not just go and complete a large number of steps, but explain to the user what needs to be done. We should collect data from Access and Publisher to see how often cue cards are used and how effective they are. It is clear that reviewers notice and write about cue cards, which may be reason enough to increase their use. Since cue cards are essentially content authoring, we should coordinate between applications and be sure that there is cross application leverage of cue card scenarios and material.

On-line documentation

For Chicago I would like to see us do two novel things with on-line documentation. First, we should be sure that our CD ROM versions of the documentation are high quality and are as pleasing to look at as printed books, and have as many links and graphical elements as our current WinHelp documentation. The Chicago Office documentation should be geared towards using the applications together, rather than just bundling all the books in one box. This will require teams to author for on-line and understand the full suite of applications. Word 7 is investigating the use of Word as a primary authoring and viewing environment for online documentation.

Second, I think we should be able to do more with VBA and the help system. For example, if I look up in help how to insert a page break, I think there should be a small fragment of VBA code that I can execute right away that will insert the page break. In general, I would view the VBA content of help as additional Wizards that we don't include in the general Wizard user-interface (to avoid overwhelming the user).

Compound files (DocFiles)

I do not think we are being clever enough in how we maximize the use of compound files. Word 6 will put embedded graphics in separate streams. All of our applications should take advantage of this and be able to Insert Picture from a Word document using this. The Shell offers a great opportunity to open up the hierarchy within the compound file. In the NeXT system, there is a analogous notion to compound files (though it is just a fancy directory) and within the browser it is possible to expand into that hierarchy with a special command. We should think about implementing such a command in the Chicago shell.

Similarly, we have done a good job in standardizing the summary properties for a document and storing these in a standard compound file stream. We need to find nice ways of using this information across applications, as Lotus does with Application Field Exchange, and within the Shell.

Database

On the system side of this issue is the fact that Chicago has the requirements for a small foot-print ISAM engine mostly for the registration database for OLE and PnP. It might be interesting to expose this as a system supported local engine. However, since we are making the Jet engine available with VB and VC++, the need for this is diminished. Before we decide to expose such an ISAM as a system API (Win32) then we need to determine if any Microsoft applications would have a use for the ISAM. It is not reasonable for Chicago to ship with Jet because of the on disk size of the engine.

With regards to tools and applications, we need to continue to push forward with VBA, Access, and VC++ integration. Access does not support any custom controls, which is a major shortcoming for programmability. Currently, Access 2.0 is supposed to ship by the end of the year, even though Access 1.1 just shipped. We should investigate the amount of work required to support either OLE 2 controls, if the timing is right, or VBXs. If the impact on the schedule is not a disaster or if it is decided that the releases are too close together anyway, custom control support should be added.

Visual C++ will be adding database support in an interim release. The product will ship with the Jet engine, just like Excel. There will also be Wizard support for developing applications in a manner similar to the bound controls of VB, but tailored for the C++ and MFC programmer.

Tools

By tools, I am referring to Visual Basic, Visual C++, and to some extent Access. There are two key issues with tools. First, how well will our tools fit into the Chicago Office environment. I believe that our tools will be able to incorporate most of the features mentioned in the *Points of Light* document without any trouble. Our tools, however, can do much more in the area of leveraging VBA, compound files, and most of all the Shell. Both VB and VC++ (and Access) currently have proprietary project management user-interfaces, which are essentially a folder in the Chicago Shell. These tools should look at integrating at least browsing functionality into a Shell extension. Similarly, these applications should consider using compound files (for VC++ this will probably be unreasonable because of the constant regeneration of .OBJ/.EXE files, but it might be more reasonable for VB/VBA to consider).

Second, we need to improve the support in our tools for Chicago specific features in terms of developing Chicago applications. VC++ will support all of the standard Chicago user-interface, OLE 2, compound files, and all of the points of light in the MFC 3.0 application framework.[3] Visual Basic should, of course, offer the same level of support. Currently, VB already supports OLE 2 and automation. An issue that continues to cause problems is the fact that VB includes its own implementation of standard Windows controls in the VB runtime. This makes it difficult for Basic applications to track the look and feel and behavior of standard Windows idioms. We need to have some architecture whereby VB applications and runtimes do not need to be revised with every release of the system. This is especially true, if we plan on having VBA on both NT and Chicago.

Currently only Access and to some degree VC++ have application-like Wizards, and only Access has cue cards. VB 3.0 included, however, an application setup Wizard, but it was not very well integrated. Our tools need to think in terms of adding Wizards that will look more familiar to users of our applications. This will become especially important as we look to integrating our productivity applications into complete solutions that use our development tools.

As we start to use development tools to drive application sales we need to be more aggressive at incorporating our applications in the sample programs and demonstrations of our tools. Today this can really only be done with VB 3 using OLE automation. If we truly believe that VBA will tie all of our products together, then we need to invest in compelling demonstration scenarios to get customers, particularly corporate developers, to believe this.

Our tools will be able to write some great *add-in* functionality for our applications, yet we do not have a common add-in architecture. Today writing an add-in requires learning the architecture for each application, and they are all different. This is something that VBA needs to define, and applications need to buy into.

Forms and Controls

Underlying our improved tools support will be a shared forms architecture for Chicago applications. This architecture is currently being designed. The plan is that all applications will use this architecture either directly, by incorporating code using these forms, or indirectly, by using VBA. The main design point of these controls is to implement them as OLE 2 lightweight in-proc servers, and enhance OLE 2 with an event architecture. There is still the problem of having a generic container along the lines of the Windows dialog manager that can contain these controls. There are no plans for the dialog manager to be modified. The VBA container, IForms, will be the container used within most applications.

Applications should ship customizable versions of all of the standard dialogs that use this forms architecture. Although the performance will be inferior to built-in dialogs, especially if SDM is used, users will still be able to customize the dialogs as needed. For example, if an VBA version of the File Open dialog is provided, then a Solution Provider or end-user could remove certain characteristics that are not appropriate for a given use of the application, such as the directory location or network options.

AFX will be implementing a Control Wizard that will speed the development of these controls and hopefully make it close to VBX in terms of ease of authorship. The performance of these forms and controls is an area of concern. The current implementation of a variant of OLE controls, Cairo's component forms, calls for the use of windowless controls. This is essentially implementing a light weight windowing system for our controls.

---

[3] An issue that this raises is discussed at the end of this memo. If there are too many Chicago-specific Windows API changes, then this will make the support for seamless Chicago/NT portability using MFC problematic.

Window management / application model

Our applications are not going to move away from MDI for the Chicago release. The Chicago Shell, however, is going to be an SDI application. This introduces a few problems that we need to work through. First, we need to improve the MDI window management functionality, which means improving the MDI API. There are a number of things people have been asking for, some of which Cairo has already developed. MDI is a model programmers use and understand today and we need to continue to support it.

Because the Shell is SDI, we will need to work through two key problems with such an architecture. First, we need to invent some pardigm to control the window proliferation problems. The Macintosh has always had problems in this area. Before System 7, many people wrote INITs (Mac TSRs) to add all sorts of bells and whitles to the Finder for better Window management. System 7 introduced a subset of that functionality. The Chicago Shell needs to do some competitive research and learn about the Finder window management problems before trying to solve this on their own. The second major SDI problem has to do with sharing user-interface elements. Unlike the Mac we do not have a menu bar at the top of the screen for all applications to share (another user-interface problem). This becomes especially interesting given that we have gone to great lengths to have similar top level menu structure in our applications. Having redundant File, Edit, Tools, View, Inert menus on every window is problematic.

The possible migration of our applications from MDI to SDI will need to be addressed in Cairo. I do not think we will be able to provide a seamless SDI implementation that usese the MDI API; in other words, new code will need to be written and major application restructuring will need to take place.

Dial-up Services

If all goes well, Chicago will have a dial-in registration service that ties into Microsoft's dial-up venture. Applications should be thinking of clever ways to take advantage of the information we will have on the user to provide a custom *Upgrade Your World* service. For example, the first time a user pulls up the Format Font dialog, the application could detect a small number of fonts and suggest dialing up and purchasing the FontPack.

Process

Most of the work and burden for integration falls on the Word and Excel teams. They are in the strongest position to undertake the development effort. If we develop some of the features discussed below in an intelligent manner, then we should be able to enhance the cross-application, and cross Windows platform, synergy of Access, PowerPoint, Project, as well as our development tools (VB, VC++).

The timing and critical nature of Chicago will make it imperative that we become more efficient at adopting shared interfaces and implementations. IDG has the responsibility to review and drive a number of these issues and has made significant progress with Word 6, Excel 5, and OLE 2. For Chicago, however, I think we need to be more proactive. Each Office application should have at least one program manager, possibly the same one(s) working with IDG today, devoted to maximizing interoperability scenarios. This person(s) should be technical enough to understand the implementation costs of scenarios. This group of program managers should meet regularly and coordinate efforts across applications and with the Interoperability group. I would also recommend that developers be assigned to work on the shared code as indicated above. It is critical for the applications developers to work directly with Chicago developers on features, such as the Explorer, that span applications needs.

I would propose that we enumerate the interesting user-interface and application features that we wish our applications to use and develop them in an MSAPPS.DLL. This effort would be shared from the

beginning, staffed with developers from both Word, Excel, and possibly the system (depending on how much we wish to keep these features to our application's advantage). The design of these components would be driven by the accomplishments made by IDG and the DAD program managers as for Word 6 and Excel 5. New features would be specified following the same process, though with a shared implementation there is a near-zero chance for variance between applications. It is important to use shared developers so the variations in host applications can be taken into account (i.e. the Excel layer, WLM, etc.) Yes, this is amazingly difficult and applications can give many reasons why this is not possible, but it is so critical to the success of Chicago and Chicago Office that is must be done.

In order for this to truly work, it is imperative that this shared code initiative be part of a controlled system. If a group needs a status bar, then it must use the Chicago (or MSAPPS) control. If the group decides not to, then it must somehow have extra resources and those resources must be removed. If the problem is one of features, then the shared code people must add those features, or a mechanism for the feature to be extended. Alternatively in a more positive light, groups could also be rewarded for using shared code.

Appendix: API Issues

An issue that has not been receiving much detailed attention is the additional APIs being defined in Chicago. Chicago is adding new APIs to USER and GDI, and modifying existing APIs in USER and GDI. These changes will not be reflected in the Windows NT Win32 subsystem when Chicago is released. In addition, the WLM layer is being designed around the official Win32s API (as published today) and has not taken into account the extensions. WLM will ship before Chicago, yet the time frame between that ship date and the next WLM release is tight (perhaps 6 months at most). In addition, the Microsoft Foundation Classes, which have as a goal to have recompile portability between NT and Windows 3.1, will have to reconcile the differences in APIs, which may mean reducing the amount of Chicago specific support or requiring customers to have conditional code in their applications.

The Chicago changes fall into two categories: completely new APIs and functionality, and modifications to existing APIs. I would suggest as a requirement that completely new APIs and functionality all be implemented in 32 bits, or at least 16/32 code that is verified to function on NT. This code could then be added to NT by just installing the DLL. Our applications could even ship these DLLs. The risk here is that in order to avoid dependency on these features and APIs, our applications, or even other ISVs, will simply reinvent the Chicago code themselves. Then in order to avoid testing issues, that code will get used on both Chicago and NT, thus removing Systems dependencies from the application.

The modifications to existing APIs/subsystems that Chicago is working on are important and should get done. The only advice I can offer, is that the changes need to be extremely well documented and tracked. The NT subsystem and WLM both have USER code in them that will need to integrate these changes. Even subtle changes in behavior, such as optimizing message ordering or changing the style of built-in controls, must be clearly documented for those groups that regularly have access to the Windows source code.

The problems, both PR and technical, that will be created by even a six month gap in Chicago and Cairo are very real. I think we are going to confuse the marketplace and the press. We have spent a great deal of effort pushing the Win32s API, and people almost understand it today. As soon as we release code that breaks that model, we will receive a great deal of criticism. It is almost like we abandoned our dream of a single Windows API.