Microsoft Word For OS/2
Development Postmortem

Richard A. Saada
January 2, 1991

What We Did Right:

Providing Support for OS/2 specific issues

PM Graphics Formats (Metafiles/DIBs)

We added support to PM Word for both PM Metafiles and Device Independent Bitmaps (DIBs). While WinWord had support for Windows metafiles, PM Word supports both the Windows and PM formats. DIB support was also added, since this is the only bitmap format available under PM. This included support for bitmaps > 64K, not available in WinWord 1.0 .

Task Switching

The WinWord code was mainly designed for Windows 2.x, and so was lacking code to correctly support Ctrl-Esc task switching. This code was added for the PM Version, and many problems with Alt-Tab and Alt-Esc task switching were fixed as well. There are still problems with the code due to differences between the Windows and PM models, but PM Word is a much better behaved app in a multitasking environment that WinWord is.

HPFS

We provided adequate support for HPFS partitions within PM Word. The original OPUS code was very tied to the DOS filename specification. We succeeded in tracking all the places that needed modification to allow support of pathnames that were > 64 characters, and filenames that were not 8.3 in form. We were limited in what we could support by the necessity to keep the length of the full filename and pathname strings in a byte. In addition, the filename code was too distributed to allow UNC support. Changing either of these would have required a major redesign of the code, and was deemed too costly for version 1.1. Never the less, the support provided is a vast improvement over what is available under DOS.

Extended Attribute Support

We provided basic support for EA's under PM. In addition to the TYPE EA, which most apps support, we also store all our document summary information in the EAs for each document. The File Find function uses the EA's if they are available, rather than opening each file and parsing the document structure to find the internal copy. Maintaining them in EA's also allows for an external app to do searches on PM Word documents via the EA's, without knowledge of our internal file format. To protect against EA loss on DOS systems, an internal copy of the summary information is still maintained.

Solving Fractional Character Width Problems

There is a major difference in the PM and Windows models that caused us a large amount of grief. Under Windows, the API to get character widths returns integers in device units. If a device has linearly scaled fonts (such as the Postscript printer), the fractional values are transformed into integers before being returned to the app. The device drivers then uses these same integer values for output, so that the App can predict where text will fall. Under the PM model, the API also returns character widths in integer device units. The device drivers, however, go ahead and use the fractional information in outputting the text. This makes justifying text with the WinWord code impossible. We ran into this problem initially with the Postscript driver. After much discussion (some of it rather heated), we were able to convince the systems group to provide the escape required to make the Postscript driver use the integer widths returned to the app. We were aided in this by the fact that any other Porthole SMK or BCL app that used the Postscript printer would expect this behavior.

Then 1.3, with ATM (Adobe Type Manager), appeared on the horizon. This reopened the issue on a global basis, since the ATM outline fonts also had fractional character widths, and were available both on screen and for most printers. The testers did some preliminary work with ATM, and it quickly became apparent that our current behavior was unacceptable. Our justification was ragged, double

underline and strikethrough were misaligned, and there were blank gaps between runs of characters. To fix this, we needed to make PM Word understand fractional character widths. This was a two part problem. First, we had to find a way to get the fractional widths from the system, since the APIs provided only return integers. Second, we had to modify the FormatLine code of Word to use the fractional information, preferably changing as little as possible. This was a high pressure problem that we had to fix in short order.

The first part was the hardest to accomplish. Paul Klinger of the Porthole group and Greg Hitchcock of the PM group were very helpful in getting around the API limitations. The final solutions was for us to get the character widths of a font whose height was the size of the digitizing grid used in creating the font. This would return the designed widths as integers with no error. We could then scale these widths to the desired point size, maintaining more accuracy than we could get from the system. The second part was actually fairly easy. We decided to port some code over from Mac Word, which already handled fractional widths, and this went it smoothly. In the end, we managed to get the code in and working in two weeks, with very few bugs.

Providing our own Porthole Support

As will be discussed later on in this document, part way through the project we lost most of our external support for the Porthole libraries. As a result, the PM Word team became responsible for tracking and fixing bugs in the Porthole libraries ourselves. We succeeded in learning the code in a short amount of time, and managed to successfully fix the bugs that arose in the last four months of the project.

Tracking OS/2 Problems

As will be discussed later on in this document, OS/2 has not been a stable platform to develop for. A large part of our time during the last 6 months of the project was spent chasing bugs in the system. The development team did a good job in tracking bugs into the system, and at the very least identifying the component causing the problem. In many cases we were able to provide the line of code causing the bug and a suggested fix with the bug report. Given the size and complexity of the OS/2 code, this was a major achievement.

Internal Picture Handling

A large chunk of the internal picture code was redesigned and improved for PM Word. The highlights of this were:

Bitmap Cache

We wrote a set of functions to cache metafiles into bitmaps, to increase the speed of our picture display. Since all but the simplest metafiles take longer to display that a bitblt to the screen, this is a clear and substantial win. During typing, a picture on the same line is shown as just the frame. As soon as typing stops, however, the picture is immediately restored. This code has also been back-ported to windows for the 1.1A release of WinWord.

Render To Clipboard

Import field metafiles, derived from graphics converters, could not be rendered to the clipboard or DDE under WinWord. If you did a copy, no other app could paste them. This was because the metafiles did not contain enough information to be displayed on their own; additional information that we stored in our internal picture structure was required. We changed the insertion process for PM Word so that the internal information was used to build new metafile records on the fly, which were inserted into the metafile. Since the resulting metafile could stand on its own, we were able to make it available to the clipboard and DDE.

Hand Building Metafiles

We developed methods to build Windows metafiles by hand. This allowed us to make DIBs from PM Word displayable in WinWord. By building a Windows metafile around a DIB bitblt record, we allow WinWord to use the Win 3 metafile code to display the DIB, even though WinWord has no DIB code itself.

What We Did Wrong (or could have done better):

Simple Port Philosophy versus a Real Project.

From the start this was an Ad Hoc project. Many of the normal procedures and processes, such as consistent Program Management, realistic scheduling and milestones, and sufficient resources, were missing, so it ended up being run on a "Management by Crises" methodology.

Changing expectations about the amount of work required.

I don't think anyone had any concept of how much work would eventually be involved in pulling this project together. When we first decided to do this project, it was originally planned to be a true port of WinWord to OS/2. Before work actually began, however, the decision was made to use the WinWord sources via Porthole. Whether or not this was a good decision is open to debate. Had the system and the Porthole libraries been stable and complete, then it would have been a definite win. As it was, it took us approximately 17 months of work to complete the project (calculating SDE months is difficult due to the constant changes in personnel and their time commitment to the project). Given the state of the OS, I don't think a true PM Port could have been done in that time. It would have required extensive changes to the WinWord sources, which were the only stable part of this project.

Once the decision was made to use Porthole, however, the nature of the project changed completely. Rather than doing most of the work ourselves, we would only be responsible for making the OS/2 enhancements to the WinWord code. For making the Windows code we wrote or inherited run on OS/2, we were entirely dependent on the Porthole Group. If they did their job as promised, our task looked to be fairly straightforward and the project would be speedily completed. This turned out to be a very big if, as the task they had set out to do became more and more difficult the longer they looked at it.

The Development team ended up spending a lot of time being testers for the Porthole group. Later in the project, we had to assume development responsibility for our version of the Porthole layer as well, and this increased the workload. In addition, the changing requirements for what builds of the OS we had to support took their toll. Just as 1.21 finally began to stabilize, 1.3 appeared on the horizon. Despite the best efforts of the systems group to kill it, it began to look like it would be the dominant OS/2 in the marketplace, and thus essential for us to support. Once again we had to reevaluate the amount of work we had to do.

Lack of Program Management

Early on in the project, Kornel Marton was officially responsible for PM Word. Other responsibilities, such as WinWord 1.1, and later WinWord 2.0, made it difficult for him to give PM Word the attention it really needed. The last part of the project was managed by Chase Franklin. For the most part this provided much more consistent guidance, but he was also pulled off to work on other projects, such as the WinWord working model.

Scheduling

This was a mess. Because of the "quick port" philosophy, we didn't have a real schedule for most of the implementation phase. Quick estimates from the developers were sent around via email, but a real task list using the Excel scheduling macros wasn't done, since it didn't seem necessary. Towards the end of the implementation phase, CBT put its foot down and demanded one. We then did a schedule for the remaining tasks we knew about at that time, and used it until we were "code complete" in February. At this point we abandoned using the schedule, since we expected to be just doing bug fixes from then onwards.

Code Complete Milestone was Bogus.

We declared ourselves Code Complete in February of 1990. At this point, all the OS/2 specific changes we knew about were done. This turned out to be a completely bogus claim. Huge amounts of work were done after we were supposedly code complete. This happened largely because there was a continuous stream of unexpected problems that arose between then and ship. These included:

1. Switching from CP 850 back to ANSI
2. Modifying Format Line to handle fractional character widths
3. Most of the PM Metafile code
4. Graphics converter code
5. Bitmap Cache
6. DIB support in porthole
7. International keyboard support in porthole
8. Font mapper changes in porthole.

and more. Even the bitmap for the About Box didn't appear until about two months before we shipped, and this became a running joke in our leads meetings. I don't really know if we could have completely solved this problem. Certainly some of the above, such as items 3 and 4, should have been scheduled. Items 2, and 6-8 were problems we had to fix as we discovered them, and 5 was a major optimization we decided to do fairly late in the game. In any case, a more careful analysis of the possible problems would have been beneficial in producing a more complete schedule of tasks for the implementation phase.

ANSI/CP850 Switch

WinWord, being a Windows app, runs in ANSI. Early on we decided we wanted to be a Code Page 850 app, as we were running in the PM environment. This turned out to be a mistake. First, it required us to run all WinWord documents we wrote through an ANSI/CP850 converter when we opened them. It also caused problems with support of publishing characters (which don't exist in CP850), and with sorting International characters. We decided at last to go back to ANSI. This meant we could read WinWord documents much easier, but that text imported from the clipboard would be translated. Unfortunately, OS/2 is very inconsistent in its support of ANSI (CP 1004). Many base functions (such as for getting sort tables) fail for ANSI, and some parts of the system (such as the FontEdit utility) won't accept 1004 as a valid Code Page. The most blatant case is the IBM 4019 laser printer, which is apparently hardware restricted to CP850. We were finally forced to drop support for this printer (except via Postscript Emulation Mode) because of this. Despite these problems, I believe going with ANSI was the right decision. If we had done more research into how hard it would be to switch to CP850 we might have just stuck with ANSI and saved ourselves some headaches.

PM Metafile Handling Redesigns

The Porthole group redesigned how a Windows app would have access to PM Metafiles several times. Each time we were required to rewrite our code to accommodate the new design. We would have to find all its bugs, and then we'd find some problem that forced them to redo it again. This should probably have been thought out better by both sides, and we should have taken a more active role in designing how this worked.

Fractional Character Width Problems

While we did a very good job of handling this problem once we had too, we should have identified it sooner. Fractional Widths fonts were actually available under 1.21 as well, in the form of Engine Outline fonts. While these were ugly enough that they were rarely used, they did exist. The early problems with the Postscript driver also pointed out this problem, but we opted for the easy solution of getting the driver to emulate the Windows model. We knew at that time that we were likely to have problems with future drivers (such as the PCL 5 driver), but we thought it was too late in the project to try and do the real fix. As it turned out, ATM on OS/2 1.3 forced us to change our minds on this.

There is also some question of whether the solution we chose was actually the best one. We discovered later (the week before we shipped), that we might have been able to solve the problem without really handling the fractional widths. The real issue at hand was not that the fonts had fractional widths, but that the driver was not doing what we predicted in our layout code. We used integer widths to format, the system used fractional widths to do the output. This could have been fixed in two ways, either by making our layout code correctly predict the system's behavior, or by forcing the system to behave the way we predicted. We chose to fix out layout code, which while it may have been the "right thing" to do, was not the easiest solution. The ExtTextOut API does provide for the application to specify the width of every character in the string at output time. We could, I believe, have modified our code to pass the integer widths we wanted the system to use on every output call, and thus forced it to match our layout. This would have required us to verified that every output call in the program correctly passed widths (there are places that don't), and would have resulted in different output. It would probably have been easier to implement, however. We didn't bother to explore this fully, since we had already solved the problem the other way.

Graphics Converters

There were several problems with our pulling together graphics converters for PM Word. The first is that we didn't get started on sorting out what we were going to do until far too late in the project. We should have had a plan as to where we going to get them from far earlier, and gotten the ball rolling.

As a result, we didn't have ANY working converters until the very end of the project, the last month or two before ship. This, of course made it very hard to test our converter interface.
The interface itself was another issue. Our Windows converter interface is based on the Aldus converter specification, so that we can share their graphics converters. For a long time we misunderstood what Aldus intended to do under PM, and this resulted in a lot of confusion on our part. The confusion was compounded by the general lack of knowledge about how PM Metafiles really worked, both in our group and in systems. When we finally found out what Aldus really intended, we decided on just modifying the Windows interface to make it PM compatible, but we ended up going through several iterations before we finally achieved a workable specification we could give to the contractors doing the converters.

Trying to support UNC paths

Early on in the project the decision was made to not try and support UNC paths, because it would be too costly to implement. Part way through last summer, it was discovered that WinWord did partially support them, and that some code added for PM was blocking it. I let myself be convinced that we should try and add real support for UNC, even though it was late in the project. This should not even have been attempted. As I had originally thought, too many places in the code were affected, and it broke the file system model in too many ways. After one release with lots of bugs, we pulled the feature and removed the code.

Setup Release Problems

There were a lot of problems discovered after we started building release disks that should have been found far earlier. The release mechanism for setup was also pretty ad hoc, which made life difficult for the testers trying to do quality assurance on our release disks. The problems we found right at the end were:

Tech Reference not being converted

The Tech Reference document that ships with the product was found to be the unmodified Windows version three days before we shipped. This file just fell through the cracks, and no one noticed until the end.

Clipboard

The Clipboard applet we shipped to provide missing system functionality could have been cleaned up with some work. It didn't have any accelerators and the internal code needed some work. This was a low priority piece, but just adding accelerators would have been an improvement. Several people didn't realize we actually planned to ship this, so it never got done.

Prev.fon weight problem

The preview font that we had on the disks was found to not match what some of the testers were using. The weight class of the font had been modified to match the OS/2 Standard (500) rather than the Windows Standard (400), and this hadn't been completely propagated.

Downloadable Symbol Font problems.

There was confusion until right at the end about whether we would include the downloadable laserjet symbol font that WinWord shipped with PM Word. As a result, they never got tested. When we did try them, the week before shipping, they turned out to have several problems. We solved the ones we could, and filed PTR's for the problems in the driver.

These should all have been handled months earlier, however, and would have been if we had started building disk sets earlier. Even though it is known that the disk sets will change and be rearranged, it is good to put a first pass at them together early on so everyone is clear just what we are planning on shipping.

International Problems

Division of responsibility between International and US with respect to who would do work for the international versions was unclear. How much of this work had to be done for the US version was also unclear early on. Whether some of this work could be done by international, or could be done after we shipped US, was something we didn't figure out until fairly late. For WinWord, international had a developer working in our building doing coding for their problems. This didn't happen for PM Word, so we ended up doing unexpected work for them. This hit several areas:

Keyboard Issues

Support for foreign keyboards was broken in Porthole. The code that we had worked only for US keyboards, and the systems team had no intention to fix it in time for our ship date. With a lot of yelling we were able to get some help from them, but for the most part we had to fix this ourselves.

Who would fix international accelerators that were broken in WinWord, and others that were broken in PM Word? Some of the WinWord code made assumptions about the positioning of certain glyphs, '*' for example was assumed to be Shift '8'. On some keyboard this is not true, and thus the accelerator keys were wrong. This was broken in WinWord as well.

Conversions

We had a leftover WinWord bug that was discovered for PM Word: We were missing a call to AnsiToOem during our file conversion process. This caused any document with an upper 128 character to not be convertible. Who was responsible for testing this?

Thesaurus

International uses a different thesaurus than the US version. Who was responsible for porting this to OS/2? I feel they should have had a developer to do it, but we ended up doing it for them.

We realize, of course, that international is a major section of our business. This is especially true for PM Word, since OS/2 has a much higher acceptance in Europe than here. Just how much work the US development team has to do for International, however, is something I feel needs to be looked at more closely. The International team set very aggressive goals as to how soon after US they wanted to ship various versions. On the other hand, they didn't provide much in the way of development support for solving issues that arose. If the US team is going to be responsible for this, then we need to know to plan on the extra workload. Time for solving their difficulties can be scheduled in if it is planned in advance, but then management needs to agree to the hit on the US date to bring in the Foreign dates.

Things we had to deal with:

OS/2:

General Instability

The perils of developing for a moving platform took their toll. Both development and testing spent days upgrading to current builds of the OS, many of which were actually less stable than their predecessors.

1.21 (Sloop)

We actually had demoable code for Comdex in Fall of 1989. At that point, IBM had been shipping 1.2 build 127 for several months, but we required the current 156 build to run successfully. We knew we had to wait for the MS version of 1.2 to ship, since the IBM version had too many blocking bugs for us to support. According to Systems, 1.2 was to ship by the end of the year, and we planned to ship in February of 1990. Despite build 164 being declared "Golden" at the end of the year, it wasn't until Summer that 1.21 build 187 finally shipped. Even then, the printer drivers that went out with 1.21 were so riddled with bugs that we knew we couldn't support them. The systems group promised that they would do a driver update in the fall with drivers that fixed all our ship issue bugs, but that still has not happened.

1.3 (Cutter)

The main reason it hasn't happened is the appearance of OS/2 1.3. We had finally managed to get a stable platform to work from (build 187), and were just updating printer drivers, when we started hearing rumors about 1.3. For weeks, the systems group told us that 1.3 would not be an issue, and that it would be killed. Unfortunately, that didn't happen, and we were stuck with deciding on whether to support 1.3 or not. For a long time we didn't think 1.3 would be an important platform, since IBM wasn't going to do an EE version, and not many major accounts would run the SE. Then, we were told: Yes, IBM was doing EE, and they planned on updating all their accounts to 1.3. This made it an essential platform to support. Since IBM was the only OEM shipping OS/2, 1.3 would be our entire market until the rest of the players in the computer industry caught up. Once MS decided it was going to OEM 1.3, it became our target OS, even though we still claim to support 1.21 (if you can get the latest drivers from the driver update that hasn't happened yet).

Supporting 1.3 was a major hassle. The early builds we got were not very stable, despite the claims that they had picked up all the MS 1.21 bug fixes. We were once again developing on a moving platform, and 1.3 also brought with it the problems of ATM. In addition, since all the development of 1.3 was being done down at Boca, we had a very difficult time getting resolutions to our problems. We eventually ended up getting Mike Maples to call his contacts down there so we could get a response.

2.0 (Cruiser)

Support for OS/2 2.0 was an ongoing issue. Every few weeks we'd get harassed by the systems group about why weren't trying to support OS/2 2.0 . Realistically, we were having a hard enough time supporting the buggy operating systems that were already shipping, without worrying about a buggy and unstable OS that wouldn't ship until 9-12 months after we did. I still feel this was the correct decision. If we need to rev our Porthole Dlls after Cruiser ships, we can do that. At that point we'll have a clear idea of what it required. The chance that 2.0 would break something else after we shipped made it not worth the effort to try and support their current code. At this point, it's still not clear when or if Cruiser will ever ship.

OS/2 Politics

The politics involved with developing for OS/2 were distracting and often obstructing. At various points in the project, we had to decide whether to support 1.2 or not (or whether just a certain CSD), 1.3 or not (and with or without ATM), and 2.0 or not. The legal wrangling over ATM caused us to get crippled versions of 1.3 for a month, during the critical time we were trying to support it. Not only did these versions have ATM removed, they were also missing the 1.21 outline fonts, which caused the system to run incorrectly.

Printing

Printing was by far the worst part of our problems with OS/2. This is covered more completely in the Testing Postmortem, but I'll summarize it here. From the beginning, the drivers were unbelievably buggy and unstable. This was a known problem when IBM shipped 1.2, and as a result no one could print from PM. PM Word pushed the drivers harder than any currently shipping app, yet despite this the OS/2 group refused for months to include us as part of their test plan and ship criteria. 1.21 build 187 shipped with drivers we knew had ship issue bugs for us, but we were told they would be fixed in a driver update in the fall. This still hasn't happened. Of the available drivers, only the latest Postscript and Laserjet drivers are currently usable, and even these have problems. For most of the project, the mechanisms were not really in place for us to report problems to the systems group and get resolutions. We often had to result to emailing or calling the developers responsible directly, which produced inconsistent results. For a long time, no one over in systems seemed to want to admit that printing was a fiasco and accept responsibility. Naturally this made getting anything fixed much more difficult.

Full PS/Metafile issues

PM has this bizarre concept of Full and Micro Presentation Spaces. The vast majority of the system just uses Micro PS's, but there are some things that they can't do. The most important of these is metafiles. There are some metafiles that cannot be displayed in a Micro PS. Unfortunately, there is no way to tell by looking at one if it will have this problem. Because most Windows apps wouldn't be able to handle PM metafiles, the Porthole code was designed around the use of Micro PS. Unfortunately, this meant that there was a large class of PM metafiles that PM Word was unable to display. We were finally forced to go in and rewrite the sections of Porthole involved to always give us Full PS's, despite the speed hit involved. We would have preferred to just request one just when we needed it, but the mechanisms weren't around to do this.

Communications with an unresponsive systems group.

This was an issue in several different areas. The major places where we had problems were with Printer Drivers (see above), submitting PTRs, and at times response from the Porthole Group. Getting the systems group to admit that there were driver bugs that should be fixed, and getting them fixed in time for us to ship, was an ongoing problem. It took a long time for us to actually get access to the PTR system, so we could enter our bugs directly rather than just sending email. Once we did get access this helped the situation a lot, but it should have happened from the start.

Porthole:

Porthole was such an integral part of this project that problems there often had widespread effects. These are some of the issues with Porthole that we had to deal with:

PM/Win 3 Incompatibility

As was mentioned earlier, the Porthole team set a impossible goal for itself. It's been said that while PM and Windows are similar in design, they differ in every implementation detail. This caused incompatibilities between Win 3 and PM that, while they might not affect a simple applet, can cause problems for an app as complex as PM Word.

Ini Files

One major area of incompatibility was with the ini files. Under Windows, the app is notified if events happen in the system that might affect it. Font changes and printer changes are the most common examples of this. Under PM, this isn't true. Printers can be installed or changed, and fonts can be installed or deleted, without the app being notified. In order to try and fix this, Porthole installed a system hook to monitor messages, and tried to notify us if anyone touched the ini file. While well intentioned, it had two major flaws. First, we got a lot of spurious INI change messages. One printer change could generate several messages to the app. Since we query the available fonts on receipt of one of these messages (a very slow process under OS/2), this caused major performance problems. Second, PM has this huge flaw in its system hooks. If another process calls a hook in one of your dlls, your dll won't exit until every other process that called it has ended! If something like the spooler calls your app (which is the whole reason you set the hook it the first place), your dll will never unlink! If the Porthole dlls didn't get unlinked when PM Word terminated, then the user would be unable to restart the app until they rebooted the system. Since this was obviously unacceptable, we had to remove the hook.

Quitting

Quitting was another area that had problems. The messages that were sent to the app when it was terminated via Shutdown didn't match the Windows sequence. Also, under Windows an app can stop itself from being closed, while under PM this is not designed in. This caused us several problems.

Incompleteness

In a lot of ways, the Porthole code was incomplete. Often we would track a bug into the layer, and find out that the code required to handle the request didn't exist. As an approximation, the first 50% of the code was done very fast (as can be seen by the fact that we had demoable code at Comdex in the fall of 1989). The next 30% was slower, and we did the last 20% ourselves as we discovered what they hadn't done. We ended up having to write the clipboard DIB support, AnsiToOem conversion, and the International Keyboard support ourselves. We also discovered areas of the code where error checking was non-existant, and had to rewrite that as well.

Support

The Porthole group fluctuated a lot in their responsiveness. Early on, when the Porthole code was just being put together, they were very good about getting things fixed. The nature of the bugs we found at that stage made them important for all Porthole apps. After we split sources, Rick Powell and Rao Remalla were in charge of our version of the code, and they were good about taking care of issues that affected us. During one of our resource crunches, however, we ran out of bugs for them to fix. We knew there were more, but lacked the testing resources to find them. At this point, Rick and Rao moved onto other projects, and the PM Word team took over support for our version of Porthole. After they left, we often had a hard time getting help from the BCL team. Even though a lot of the problems we found affected all Porthole apps, their priorities often didn't mesh with ours. Sometimes we were able to get help quickly, but in some cases (like the international keyboard support), we ended up doing it ourselves. Just where we fit into the scheme of things left something to be desired.

Lack of Knowledge about OS/2 in house.

There are huge gaps in the in house knowledge about OS/2. Finding the right person to ask questions of was difficult or impossible. With Windows, you can ask intricate questions of the developers and get an answer. With OS/2, you often had to send mail to someone you didn't know in Boca Raton,

Florida or Hursely, England, and if you were lucky you got a reply. Usually it was "I don't know", "Ask someone else", or answered a different question than the one you asked. Metafiles were a prime example of this. Because of the JDA, the work to design, implement, and test metafiles was done at IBM Hursely. As a result, no one at MS had any idea how the metafile code really worked, what the file format was like, or what the limitations were.

Access to the OS/2 sources was essential.

In general, the OS/2 documentation isn't as good as we'd like. The API descriptions are generally okay, but information on how to use them is lacking. Responsiveness about possible bugs was lousy unless we could prove the bug was in their code. This required debugging through the system, printer drivers, etc. until we could isolate the exact cause of the bug. We spent a lot of time understanding how the system internals worked so we could find their bugs. This would have been impossible to do without access to the system sources. On a mature and stable OS, where you can depend on the API's to do what you expect (and the expectations are well understood), I can see how access to the system sources is unnecessary. With OS/2, I don't see how it would be possible to successfully write a major app without it.

Resource Allocation Problems

This project suffered from resource problems from the very start. It wasn't until after WinWord 1.0 shipped that we started to get a real development team, but even then the team was constantly in flux as people were pulled onto higher priority projects. While I understand the relative importance of the projects involved, and agree with the decisions made, it is obvious that these events had a serious impact to both the morale and the schedule of this project. The biggest hit to our team was WinWord 1.1. When the decision was made to do this update, all but two of our testers were pulled off PM Word to work on it. This had a strong negative impact on the project, including causing Rick Powell and Rao Remalla to move onto other projects for lack of work. In addition, I was pulled off to do the autoswitching 3D visuals for 1.1 . While I was the best person to do the job, having done the 3D visuals for PM Word, the fact that they pulled the development lead off to work on another project says something about the priority placed on PM Word. We also lost Bob Zawalich to the Japanese Word Processing Project. He was in Japan for several weeks during the summer, and also spend half his time working for them for the remainder of the project. Phillip Garding was pulled off to work on the setup for the Windows Office, a project that dragged on and on for weeks.

Difficulty Determining the True Source of a Bug

PM Word was by its nature a very complex app to debug. A large and complicated Windows app, running through a developing translation layer, on top of a developing OS. Is the bug in the program, Porthole, or the system? Two of the three were unstable for most of the project. Only the Opus code could be considered at all reliable, and we found a lot of bugs there too.

Conclusions

Word for OS/2 was not the most enjoyable project to work on. The team was often frustrated with the lack of importance placed on the project, and with the platform we had to work with. A large part of our time was spent putting out fires, arguing with IBM and Systems, and doing a lot of mind numbing debugging. One lesson we learned is the danger of having too strong a dependency on an untested (and at that point unwritten) component over which we have little or no control. As much work as it was to maintain the Porthole dlls ourselves, at least then we knew what we were dealing with and could make intelligent decisions about how much work was to be done. We did, however, avoid the Excel trap of shipping a product before the system was in good enough shape to support us. The team motto developed at the end of the project pretty much summarizes our feelings about shipping PM Word...

... Against All Odds