Warp vs. Chicago

A Decision Maker's Guide to 32-bit Operating System Technology

IBM Personal Software Marketing

September 1994

Executive Summary

This document is designed to provide the corporate decision maker with benefits of OS/2 and important information about critical weaknesses in Microsoft's forthcoming Chicago operating system. At the heart of the discussion are key architectural, operational, and strategic flaws in the Chicago OS design and strategy - flaws that Microsoft has either downplayed or ignored in its efforts to market Chicago as the "next generation" Windows desktop platform.

For example, you'll learn:

- Why OS/2's ability to isolate individual 16-bit Windows applications into their own separate VDMs provides a level of inter-application protection that is unavailable under Windows 3.1 or Chicago.
- How this same isolation also allows OS/2 to preemptively multitask existing 16-bit Windows applications, with no impact on native application performance
- Why having a comprehensive System Object Model (SOM) is important, and how OS/2's SOM implementation acts as the "glue" to the WorkPlace Shell interface.
- Ways in which OS/2's Virtual DOS Machine implementation is more flexible than Chicago's.

Major topics include:

- Architectural flaws that compromise Chicago's stability when running 16-bit Windows applications.
- How these same flaws also limit Chicago's multitasking capabilities with a mixture of application types.
- Why the lack of a System Object Model makes the Chicago interface "fragile."
- Ways in which Chicago's DOS heritage render the product inflexible when dealing with 16-bit DOS device drivers.

At the end of each section, a direct comparison is made between the Chicago implementation of a particular subsystem or feature/function, and that of the leader in 32-bit desktop operating systems, IBM's Operating System/2.

The material is based on an in-depth analysis of Microsoft's public statements regarding Chicago's design characteristics and various presentations given at trade shows by industry consultants.

OS/2 - The Right Solution

Choosing the right operating system. In many ways it's the most important personal computer technology decision you'll make in this century. Choose wisely and you'll reap the benefits for years. Choose poorly and you may find yourself in a quagmire of under-performing software and inadequate computing power.

So just what constitutes a wise choice in today's confusing PC marketplace? Simple: the product that does the best job of preserving your existing investments while opening the door to the future. In a nutshell, the wise choice is Operating System/2.

OS/2 - The World's Most Popular 32-bit Operating System for IBM and IBM compatible PC's

Why OS/2? Because it represents the most logical upgrade path for today's PC users. OS/2 preserves your investment in 16-bit DOS and Windows applications while providing access to a new world of 32-bit, object-oriented technology.

Upgrading to OS/2 is a win-win proposition. Just ask any of the more than five-million OS/2 users - over 8 times as many users as Microsoft's current 32-bit offering, Windows NT. These are people just like you who have outgrown their existing DOS or Windows environments and who are looking for more - more power, more functionality, more stability.

With OS/2 they've found a powerful mix of backward-compatibility, 32-bit processing power, and ease of use, along with the kind of rock-solid reliability that only a mature, established operating system platform can deliver. With the release of V3, OS/2 is entering in its 3rd generation, and the product's reputation for reliability and price/performance is unmatched in the PC industry.

But what about Chicago?

This is the question that perplexes both corporate decision makers and end users alike. With all of the media hype surrounding this "next generation" of Microsoft Windows, many customers feel paralyzed when making operating system purchasing decisions. The fear of "missing-out" on Chicago is overwhelming for some.

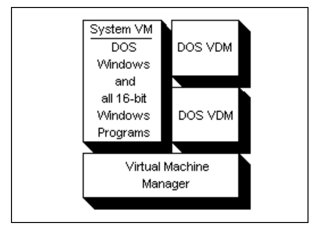
But as experience with the initial beta release of Chicago has demonstrated, Microsoft's "next generation" of Windows is far less compelling than they would lead you to believe. In fact, the core of Windows 4.0 is probably running on a PC near you: it's called Microsoft Windows 3.1.

Architecture

Chicago - Same Code, Different Packaging

"How can that be? It looks so different!"

Looks can be deceiving. While Chicago indeed sports a radically different user interface (more on that later), as you peel-away the layers of GUI and packaging you'll discover a product that looks remarkably like Windows 3.1. In fact, Chicago retains so much of its original DOS/Windows heritage that it retains the latter's most notorious operational characteristic: instability.

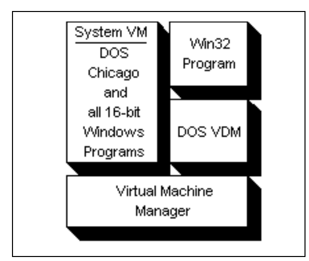


Windows 3.1:
The OS "house of cards"

For example, under Windows 3.1 all applications, as well as the operating system code itself, share a single memory address space. While such a memory management model breeds performance, it also means that an error in any single application can potentially crash the entire operating system.

This crashing phenomena is often referred to as a General Protection Fault or "GPF," and has been the bane of Windows users since version 3.0. It is because of this inherent architectural weakness that Windows 3.1 has gained a well-deserved reputation of being an unstable, unreliable operating environment.

Under Chicago, this same single address space model (referred to as the "System Virtual Machine") is retained, along with the inherent weakness of leaving key portions of the operating system code exposed to potentially buggy applications. Thus the same application failures that crashed Windows 3.1 can potentially bring down the entire Chicago operating system.



Chicago: same code,
different packaging

To their credit Microsoft has made great strides in "cleaning-up" many of the bugs in the original Windows 3.1 code while preparing it for inclusion with Chicago. However they cannot avoid the inherent architectural flaws that the Windows 3.1 single System VM model introduces. There will always remain the possibility of an errant application causing a disastrous system crash.

OS/2 - Same Code, Better Implementation

OS/2 eliminates the Single System VM stability problem by letting you run Windows applications in their own separate sessions, or "VDMs" (Virtual DOS Machines). Thus if an application fails under OS/2, the effect of the failure is limited to the individual session. Other applications, as well as the operating system itself, remain unaffected.

And by retaining much of the original Windows 3.1 code base, OS/2's environment remains highly backward compatible with Windows 3.1 applications and device drivers.

Multitasking

Chicago - a "Semi-Preemptive" Task Switcher?

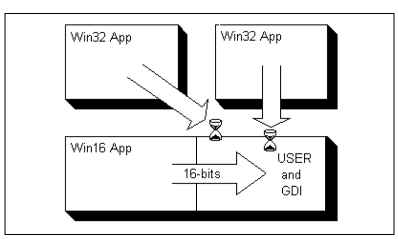
One of Microsoft's biggest selling points for Chicago has been the promise of a new breed of 32-bit Windows applications. These applications are to be preemptively multitasked by the Chicago operating system, and will have access to advanced performance enhancing techniques like multi- threading.

Let's define the difference between preemptive and cooperative multitasking. Preemption is an involuntary loss of control which the application must handle. Cooperative multitasking is where the application is given control and it is the application's responsibility to give up control so that other applications may execute.

The move to a preemptive multitasking model represents a a significant departure from Windows 3.1. Under that environment applications must "cooperate" in order for multitasking to occur. Each program "yields" to the operating system so that it can switch control of the PC's CPU to a different application (this is often referred to as "cooperative multitasking" or "task-switching").

It is a well know fact that the Windows "cooperative multitasking" model is inefficient. It also forces programmers to code their applications in a way that adds complexity and hinders performance. So it comes as no surprise that Microsoft's promise of preemptive multitasking in Chicago has been heralded as one of the new platform's most important features.

But the truth is that Microsoft isn't telling the whole story when it comes to Chicago's multitasking architecture. In reality, unless you work exclusively with 32-bit "Win32" applications, you won't reap the benefits of true preemptive multitasking.



Chicago: roadblocks on the way to preemptive multitasking

Why? Because of Chicago's heavy reliance on 16-bit, Windows 3.1-era code. Under Chicago, both 16-bit and 32-bit applications rely on 16-bit code structures that reside within the System VM - code that has been brought over from Windows 3.1.

While the "bitness" of the code itself isn't significant, the environment from which it hails is. Windows 3.1 was written as a cooperative, not preemptive, multitasking environment. When you introduce portions of its code into a preemptive setting, where more than one task may be vying for its services at any given time, the code breaks.

To safeguard against this sort of "code breakdown," Microsoft has serialized access to key portions of the Chicago infrastructure - most notably the USER (window management) and GDI (graphics device interface) subsystems. In technical terms, this is referred to as a "non-reentrant" design, meaning that only one application may execute within these modules at any given time.

While such an approach works with Win32 applications - which can be preempted at any point during their execution - it breaks down once a 16-bit Windows (Win16) application begins to execute. As it stands, currently shipping Win16 applications cannot be reliably preempted during execution. Attempting to do so while such an application is calling on a non-reentrant, 16-bit code module can cause the entire operating system to crash.

To avoid this latter scenario, and thus retain some semblance of multitasking, Microsoft has implemented a special locking mechanism. Dubbed "Win16LOCK," this mechanism denies access to the older code when a 16-bit application has called on its services. Thus only the currently running Win16 application has access to the 16-bit code - all other applications, including Win32 applications, are "blocked" from executing until the 16-bit application has finished and the environment has been made safe for the next task.

In practice, the performance hit associated with this locking phenomena is minimal when running 32-bit applications exclusively. However, when you introduce a mixture of 16 and 32-bit applications - the most likely scenario given the projected lack of available Win32 products - Win16LOCK becomes a major problem.

Application Type	**Multitasking**
Win32	Preemptive
Win16	Cooperative
Mixed Win16 and Win32	"Semi-preemptive"

Most 16-bit Windows applications are notorious for failing to yield properly under Windows 3.1, and until they do so under Chicago, all other applications will be blocked from accessing USER and/or GDI (in reality, only 50% of GDI calls are affected - but these are the most common functions so the net result is the same).

Taken as a whole, these two compromises - the serialization of subsystem access and Win16LOCK - create what would best be described as a "semi-preemptive" multitasking environment. And while the resulting "hourglass" is expected under a cooperatively multitasked environment, it seems out of place in a "next generation" Windows that supposedly "preemptively multitasks" native Win32 applications.

OS/2 - True Preemption for Better Performance

OS/2 has featured true preemptive multitasking of native applications since day one. Regardless of the mixture of application types, OS/2 can continue to smoothly multitask dozens of concurrent programs, and its reentrant subsystems allow it to service multiple concurrent requests without the overhead of a "Win16LOCK" implementation.

And thanks to its ability to run them in separate VDMs, OS/2 can also preemptively multitask existing 16-bit Windows applications which Chicago can not. Thus you can have DOS, Windows, and OS/2 applications running concurrently, side-by-side, without any performance penalties and all preemptively multitasked. This is a feature that Chicago will be unable to match without underlying architecture changes, and a welcome addition to any power-user's arsenal.

Interface

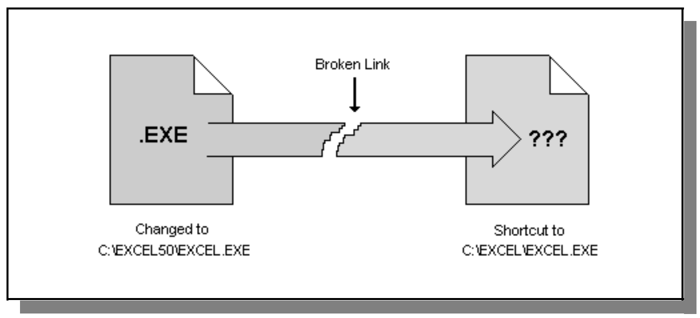
Chicago - Beauty That's Only Skin-Deep

Another major feature of Chicago, and one that has drawn considerable attention from the industry press, is its new user interface. Terms like "object-oriented" and "desktop metaphor" are often used to describe this radically different Windows look.

But as with most of Chicago's underpinnings, the actual foundation underneath the product's user interface is nothing more than an extension to what already existed in Windows 3.1. Unlike a true object-oriented environment - where links between individual objects are "live" and updated automatically - the Chicago GUI is static. "Objects" on the Chicago desktop are merely pointers to files on the disk. "Properties" for these objects are stored in .INI files (for Windows applications) or .PIF files (for DOS applications), and links between them (called "shortcuts" under Chicago) are equally static.

For example, if you create a shortcut to an executable file and place it on the Chicago desktop, then rename the original executable, the shortcut will essentially be severed. To re-establish it you'll have to re-create the shortcut from scratch.

In a true object-oriented environment, all shortcut-like links to the executable would have been updated automatically by the underlying object management model. Chicago has no such underpinnings, so links are easily broken by novice users who are unfamiliar with the crudeness of the Chicago interface.



Going hand-in-hand with Chicago's shortcut mechanism is the product's support for long file and directory names on FAT volumes. Microsoft is emphasizing Chicago's ability to automatically convert long file/directory names into 8.3 character abbreviations for compatibility with existing DOS and Windows applications. What they seem to be ignoring, however, is the fact that promoting the use of long names can be disastrous when there is no underlying object model.

Take, for example, the novice user who, upon discovering long filenames, decides to "reorganize" their hard disk. They gleefully rename directories at will, unaware that they are severing shortcut after shortcut in the process. Suddenly none of their applications work, and I/S is called in to undo the damage (which in some cases may mean reinstalling both operating system and applications).

The Chicago desktop itself is not an OLE 2.0 object. This statement in itself has no ramifications until you start understanding what type of integration is lost due to this lack of object technology. This deficiency in the product, means that an application is not well integrated with the desktop and does not inherit any of the advantages like Drag 'n' Drop support.

Heralded by Microsoft as one of Chicago's key selling points, the new Windows interface may in the end prove to be one of its biggest flaws. Without an underlying system object model to tie everything together, this new "shell" may prove to be an I/S support nightmare.

OS/2 - True Object-Orientation

OS/2's WorkPlace Shell is a true object-oriented interface. The underlying System Object Model (SOM) provides complete object-tracking so simple operations like dragging a directory to another directory won't invalidate links and other interface structures. Thus it's easier on both novices and IS support staff alike.

SOM also allows applications to fully manipulate the WorkPlace Shell interface. A good example is cc:Mail for OS/2, which uses SOM to seamlessly integrate its in/outbox interfaces with the WorkPlace Shell desktop. This level of integration isn't possible under Chicago since its shell is itself not an object.

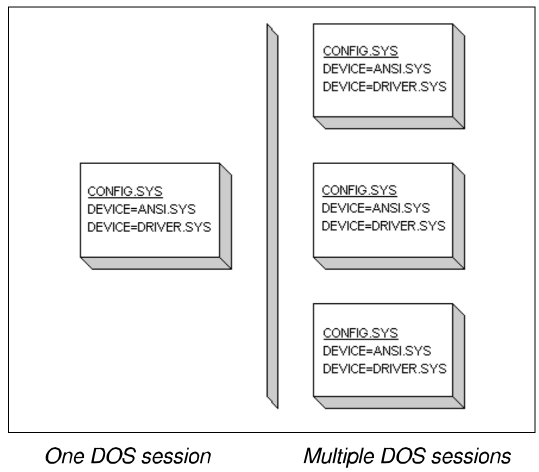
Application Support

Chicago - Still DOS After All These Years

"Chicago eliminates the need for DOS. It is a true operating system..."

This is one of the more colorful myths surrounding Microsoft's Chicago operating environment. Microsoft claims that Chicago eliminates the need for DOS - that DOS and Windows are now completely integrated and that all the old restrictions that DOS brought to the table have been eliminated.

While it is true that you will no longer have to purchase a separate DOS product in order to install and use Chicago, this in no way constitutes the eradication of DOS as a part of the Windows operating system equation. DOS is still there, lurking in the shadows. It's just been cleverly disguised by a different Windows GUI. And though much of its functionality - including file system access - has been replaced by 32-bit Chicago VxDs (Virtual Device Drivers), there are still ways in which DOS can hinder the Windows environment.



Chicago loads all the device drivers for each DOS session in conventional memory

Take real-mode device drivers, for example. Under DOS/Windows 3.1 you were forced to load all DOS device drivers at DOS boot-time via the CONFIG.SYS file. These drivers would then occupy all DOS sessions under Windows' 386 Enhanced Mode, impacting their available conventional memory and limiting the overall configurability of the Windows VDM architecture.

Chicago suffers from this very same limitation. Any real-mode DOS device drivers that you wish to access from within Chicago must be loaded via CONFIG.SYS at boot-time. Thus, if you want access to a particular resource, and this resource requires a DOS device driver, you'll be forced to pay a penalty in terms of lost conventional memory and potential compatibility problems across all Chicago VDMs.

And what about troublesome applications like games? Chicago features a special DOS session - the "Single MS-DOS Application Mode" - that allows such applications to execute unencumbered by the confines of a traditional Virtual DOS Machine (virtual I/O, video memory, etc.). What Microsoft doesn't publicize, however, is the fact that, in order to invoke this mode, you must essentially shut-down Chicago. All running applications close, and the Chicago GUI itself is paged to disk. This entire process can take up to a minute depending on the speed of the hardware in use and the number of open applications - quite a disruption, especially when you're trying to finish that last minute memo or download a large file from a host system.

OS/2 - A Better DOS than DOS (or Chicago)

OS/2 really does eliminate the need for DOS. It's VDMs are completely configurable, allowing you to create individual CONFIG.SYS and AUTOEXEC.BAT files for each DOS session. This is an important option in those situations where a single device driver or TSR configuration for all VDMs would be inadequate.

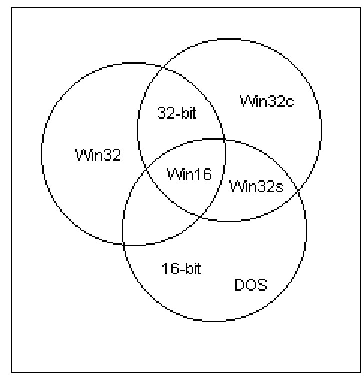
OS/2's VDMs are also highly backward-compatible and can also be configured to allow direct hardware access for applications that require it. And if an application truly refuses to run under OS/2 you can use the "dual-boot" option to run real DOS in about the same amount of time it takes you to invoke Chicago's "Single MS-DOS Application Mode."

Independent Software Vendor Commitments

Chicago: An ISV Headache

One area where Microsoft continues to be uncertain is on the subject of API standards. Independent Software Vendors (ISVs) have been fighting an uphill battle in their efforts to pin-down Microsoft's overall API strategy. This is especially true of the native Chicago API, Win32c, which is itself a subset of the full Win32 API published nearly two years ago and implemented on Windows NT.

Further exacerbating the situation is Microsoft's continual updating of the Win32c specification. New APIs emerge almost monthly, many of which extend Win32 in ways that tie applications to the Chicago platform. This has aggravated ISVs who wish to write cross-platform applications for Windows, Windows NT, and Chicago. The only way these ISV's can write cross-platform applications, because of the different APIs support, is to poll the Kernel, determine which API is available and write dual or triple path code. With the APIs still in a state of flux there is no guarantee that the multiple path code will work.



Microsoft's API Quagmire

What this means to the 32-bit operating system customer is a potential delay in the release of Chicago-compatible Win32 applications. Given the architectural limitations of Chicago's Win16 application support - especially when multitasking and stability are major considerations - lack of Win32 applications could represent a serious obstacle to the platform's widespread adoption. Chicago needs Win32 applications before it even begins to make sense as a replacement for Windows 3.1. But given the confusion and frustration in the ISV community it may be some time before we see a substantial selection of Win32 titles.

OS/2 - A Consistent Message

In contrast to Microsoft's "API du jour" strategy, IBM has stood firm on its promises to support open standards and honor ISV commitments. There is one 32-bit OS/2 Presentation Manager API for both client and server systems. Applications written to that API will work across OS/2 versions running on Intel-based PC's, and will be easily portable to more advanced implementations in the future (including OS/2 for PowerPC).

OS/2 currently boasts over 2000 native applications, all of which tap into the superior multitasking and performance of the world's most popular 32-bit operating system.

Summary

OS/2: the Right Answer

As you can see, Microsoft's Chicago operating system is long on hype and somewhat short on technology. But if you've followed their product offerings over the past few years, this revelation should really come as no surprise. Microsoft has a track record of delivering "cosmetically advanced" operating systems while ignoring the more important issues like robustness, capacity, and true object-orientation.

In contrast, IBM has a very different track record, one that speaks of commitment to open standards and listening to customer needs. This is the same company that has been developing cutting edge OS technology for mainframe and minicomputer systems since the dawn of the information age. With OS/2, IBM has laid the foundation for a truly robust, high-capacity computing environment that preserves your existing investments while opening the door to the future.

You can see the difference in areas like the OS/2 user interface. The WorkPlace Shell, in conjunction with the System Object Model (SOM), provide a truly object-oriented computing environment, one that thinks for you and doesn't break-down when you try to tap into its power. Likewise, OS/2's multitasking represents a no-compromises approach to bringing this powerful capability to the masses. From native OS/2 applications to its robust Win-OS2 VDMs, it is an operating system that can juggle your most complex tasks with ease.

So in the end, the wise choice is obvious: OS/2 has the backward compatibility you want, the stability and reliability you need, and the kind of rock-solid commitment to excellence you've come to expect from the world's number one software company, IBM. Chicago looks more and more like a warmed-over version of yesterday's technology, not the "next generation Windows" platform that Microsoft is advertising it to be.

So what about Chicago? Good question! With one foot still buried in the DOS/Windows grave, Chicago is yesterday's technology dressed-up to look like tomorrow's 32-bit OS. Why wait for an impostor? OS/2 is here today, and represents the real future in personal computer operating systems.

Appendix A: Features Chart for OS/2 and Chicago

The following charts provide a summary of OS/2 and Chicago features, including multitasking characteristics, application environments, and bundled productivity tools.

OS/2 vs. Chicago on Architecture

Feature	Warp	Warp LAN Client	Chicago
32-bit Window Management	Yes	Yes	No[1]
32-bit Graphics Subsystem	Yes	Yes	No[2]
32-bit Printing Subsystem	Yes	Yes	Yes
32-bit Multimedia Subsystem	Yes	Yes	Yes
32-bit Kernel	Yes	Yes	Yes
Demand Paged Virtual Memory	Yes	Yes	Yes
HPFS Support	Yes	Yes	No
Non-locking Input Queue[3] (Applications can keep running)	Yes	Yes	No

[1]USER is 16-bit, non-reentrant code
[2]50% of GDI calls are serviced by 16-bit, non-reentrant code
[3]WARP, new version of OS/2, has an engine that will unlock the input queue if it is locked

OS/2 vs. Chicago on Application Environments

Feature	Warp	Warp LAN Client	Chicago
16-bit OS/2 PM Applications	Yes	Yes	No
32-bit OS/2 PM Applications	Yes	Yes	No
Win32s Applications (version 1.0 & 1.1)	Yes	Yes	Yes
Preemptive Multitasking[4]	Yes	Yes	No
Win16 Application Support	Yes	Yes	Yes
Win16 Device Driver Support	Yes	Yes	Some[5]
Number of 32-bit Applications Available	2000+	2000+	0[6]

[4] See chart on multitasking comparison
[5] Windows 3.x communications drivers need to be re-written
[6] Native Chicago applications

OS/2 vs. Chicago on Multitasking Characteristics

Feature	Warp	Warp LAN Client	Chicago
Preemptive of 32-bit Applications	Yes	Yes	Yes
Preemptive of DOS Applications	Yes	Yes	Yes
Preemptive of Win16 Applications	Yes	Yes	No
Preemptive of mixed 16/32-bit Applications	Yes	Yes	No[7]
Multiple, Protected Win16 VDMs	Yes	Yes	No[8]
Crash Protection	Yes	Yes	No[9]
Preemptive Multithreading	Yes	Yes	Yes

[7] Win16LOCK prohibits access to USER and portions of GDI when a Win16 application is executing
[8] All 16-bit applications share a single address space - the System Virtual Machine (VM)
[9] Key operating system code structures (USER and GDI) share the System VM address space with 16-bit applications

OS/2 vs. Chicago on User Interface

Feature	Warp	Warp LAN Client	Chicago
Folder Work Areas	Yes	Yes	No
Integration with Operating SOM	Yes	Yes	No[10]
Launch Pad	Yes	Yes	Yes
Drag & Drop Deletion	Yes	Yes	No
Drag & Drop Faxing	Yes	Yes	Yes
Drag & Drop Access Paths (change execution paths it will still work)	Yes	Yes	No
Object Type Templates	Yes	Yes	No
Parent Folder Closing Options	Yes	Yes	No

[10]Chicago shell components are not OLE 2.01 objects

OS/2 vs. Chicago on Multimedia

Feature	Warp	Warp LAN Client	Chicago
Image Viewer	Yes	Yes	No
Photo CD Support	Yes	Yes	No
Autodesk Animation	Yes	Yes	No
Play any Audio File from Internet	Yes	Yes	No
Audio/Video Synch Manager	Yes	Yes	No
MPEG Support	Yes	Yes	Yes
32-bit Audio/Video Playback	Yes	Yes	Yes

OS/2 vs. Chicago on Bundled Applications

Feature	Warp	Warp LAN Client	Chicago
Internet Access Tools	Yes	Yes	No
· FTP	Yes	Yes	No
· Telnet	Yes	Yes	No
· Gopher	Yes	Yes	No
· Newsreader	Yes	Yes	No
· WEBExplorer	Yes	Yes	No
CompuServe Front-End	Yes	Yes	No
Word Processor	Yes	Yes	No[11]
Spreadsheet	Yes	Yes	No
Database	Yes	Yes	No
Charting	Yes	Yes	No
Report Writer	Yes	Yes	No
Electronic Mail	Yes	Yes	Yes
Image Viewer	Yes	Yes	No
FAX	Yes	Yes	Yes
Phonebook	Yes	Yes	No
Personal Information Manager	Yes	Yes	No
Sys Info	Yes	Yes	No
VideoIn	Yes	Yes	No
Video Conferencing	Yes	Yes	No

[11]Chicago comes with a simple text editor, not a word processor

Disclaimer

The information contained in this document represents the current view of IBM Corporation on the issues discussed at the date of publication. Because IBM must respond to changing market conditions, it should not be interpreted to be a commitment on the part of IBM, and IBM cannot guarantee the accuracy of any information presented after the date of publication.

This document is for informational purposes only. IBM makes NO WARRANTIES, EXPRESSED OR IMPLIED, IN THIS SUMMARY.