Chicago Feature Specification

Revision 1.00 9/30/92

Introduction

This document contains the list of new features included in the Chicago product. The feature set is described assuming Windows 3.1 as the base, thus only differences between Windows 3.1 and Chicago are shown.

This document contains a high-level description of features only, although the *Additional Information* section at the end of this document contains related specifications and technical documentation. Chicago Program Management will maintain this specification throughout the Chicago development process and redistribute it as changes warrant. Detailed feature specs will be distributed on-request and as available.

DISCLAIMER: The features described in this specification are subject to change without notice due to unforeseen implementation complications, schedule constraints, beta test feedback and usability testing. Nothing in this document should be interpreted as a contractual obligation by Microsoft to meet specific dates, testing procedures or product specifications.

This document is MICROSOFT CONFIDENTIAL

Table of Contents

HIGHLY CONFIDENTIAL

MX 2120373
CONFIDENTIAL

CMS 00013667

RBC 04068

HIGHLY CONFIDENTIAL

MX 2120374
CONFIDENTIAL

CMS 00013668

RBC 04069

HIGHLY CONFIDENTIAL

HIGHLY CONFIDENTIAL

HIGHLY CONFIDENTIAL

MX 2120382
CONFIDENTIAL

CMS 00013678

RBC 04077

HIGHLY CONFIDENTIAL

MX 2120383
CONFIDENTIAL

CMS 00013677

RBC 04078

1. COUGAR

1.1. INTRODUCTION -

This document contains the list of new features included in Cougar The feature set is described assuming DOS 5 0 and Windows 3.1 as the base. Thus only the differences between Chicago and this assumed base are called out.

This document contains a high level description of the features only. Program Management will maintain this "feature specification" set through the rest of the development process and redistribute this document as and when necessary. Detailed specs of some of these features will be written and made available as necessary.

Disclaimer: The features outlined in the specification are subject to change due to unforeseen implementation complications, beta test feedback, and usability testing. Nothing in this document should be interpreted as a contractual obligation by Microsoft to meet specific dates, testing procedures or product specifications.

1.2. OVERVIEW

AS-DOS

1.2.1. What is Cougar?

Cougar is an enhanced mode kernel for Chicago, that maintains compatibility with existing DOS 5.0 and Windows 3.1 applications, TSRs and drivers. It provides a robust and high performance base for the next version of Windows. Cougar actually provides the user with an integrated Windows and DOS environment. Cougar has its origin in the Windows 3.1 enhanced mode kernel (WIN386).

1.2.2. What are the 5 important features of Cougar?

- X Plug and Play support in the form of hardware detection and configuration and loadable device drivers (VxDs).
- Win-32s+ API support in the File system, Memory manager, Scheduler, etc., to provide the relevant primitives that are part of the Win-32 specification
- X A robust, small, and high performance protect mode kernel that maintains compatibility with existing DOS 5 and Windows 3.1 applications.
- X Smoother Multi-Tasking accomplished by a combination of multi-threaded protect mode kernel, protect mode file system and protect mode device drivers that support asynchronous I/O.
- Easier DOS application configuration and improved DOS application compatibility.

1.2.3. Other Cougar features

- ? Improved memory manager which supports sparse memory and memory mapped files, and a dynamic cache allocation scheme. Both of these allow for the more efficient use of existing memory.
- Support for multi-threaded applications, that aids smoother multi-tasking.
- 32 bit protect mode communication drivers for a higher performance communication throughput.
- An Installable File System (IFS) mechanism and a 32 bit FLAT FAT file system (VFAT) together provide better performance. The IFS also facilitates the easy addition of other file systems.
- Optional User visible file system enhancements such as last accessed date and long file names (?).
- ? Protect mode CDROM file system, for better performance.
- A consolidated DOS and Windows configuration database, that also adds more configuration properties for DOS applications (the implementation scheme has not yet been solidified): This attempts to consolidate the information that is currently spread all over the place and lends itself to easier administration.

- Protect mode Layered Device drivers to make it easy for IHVs to provide value added hardware/software (DRAGON). [PROVIDE EXAMPLES]
- PCMCIA Support
- Better SCSI support. [PROVIDE EXAMPLES]
- Integrated 32 bit print spooler (in Ring 0) for both DOS and Windows printing.
- Integrated networking with NDIS 3.0 drivers (VxDs) that provide better performance and reduce conventional memory usage.
- DOS VM improvements such as True type fonts, Cut/Paste toolbar, UNC path names and multiple parameter support for DOS commands.

1.3. REQUIREMENTS

- The following are the requirements for Chicago and they apply to Cougar as well:

- Compatibility: is assumed to be 100% with Windows 3.1 and DOS 5.0, unless specifically noted. This implies that any feature that breaks compatibility has to get specific exception. Otherwise it will be categorized as a bug.

- Performance: should be at least as good as Windows 3.1 and DOS 5.0, for a specific set of Applications. The set of applications will be defined based on input from marketing and testing.

- Robustness: of the system should be at least as robust as Windows 3.1 and DOS 5.0, when running a specific set of Apps. The set of applications will be defined base on input from marketing and testing.

- Size: of the system should allow the loading of the entire system and the running of a couple of Apps on a 386 20 MHz DX (4 Meg RAM) machine, with a response time that is at least as good as that of Windows 3.1 and DOS 5.0. The specific apps that will be used for testing will be specified by the testing group.

1.4. COUGAR FEATURES

1.4.1. Priorities definition

1=Must have for Chicago.
2=Nice to have for Chicago.
3=We will try to do this for Chicago, if we have time.
4=Don't waste time on this for Chicago.

1.4.2. DOS VM Improvements

HIGHLY CONFIDENTIAL

1.4.2.1. Toolbar for DOS Apps in a Window

Priority	Status
1	Done

- Cut and paste for character mode DOS apps that allows consistent mouse controls
- User choosable fonts
- DOS VM settings, such as exclusive, foreground, priority etc.

1.4.2.2. Better user control over conventional memory

Priority	Status
1	

MX 2120385
CONFIDENTIAL

CMS 000/3679

RBC 04080

DOS 5.0/DOS 6.0 allows for a user to load Device drivers and TSRs into UMB area but other parts of the system are loaded in the HMA or conventional memory below 640Kb. We should provide a InstallHigh, BufferHign, LastDriveHigh, FilesHign, FCBSHigh type of options for user to be able to better optimize the system memory.

1.4.2.3. Improved graphics support for Graphics DOS Applications

Priority	Status
2	

Only a limited number of Graphics modes are supported in a DOS VM. The list of supported Graphics modes will be increased to encompass the most popular graphics modes on display adapters, like support for 1024x768 mode.

Chicago VMs which run in non-standard display modes should require a VDD loaded which can restore the display in that mode. If no such VDD is loaded, then WinOldAp should warn before going into the extended mode VM (if this is known at startup), and the PC should beep if the user attempts to ALT-ENTER or ALT-TAB out of the full-screen VM. This will prevent crashes, hangs, and incorrectly drawn extended mode VMs due to trashed display registers.

[ENUMERATE THE LIST OF NEW GRAPHICS MODES THAT WILL BE SUPPORTED].

1.4.2.4. Ability to launch Windows apps from DOS command line

Priority	Status
1	Mostly Done

The ability to launch Windows applications from the command line is currently provided by other third party apps, such as WINCLI. COMMAND.COM will be enhanced to provide this feature.

1.4.2.5. Use TrueType Fonts For Scalable DOS Boxes

Priority	Status
1	

The user can choose to display TrueType fonts in a DOS Box. This will allow the user to scale the DOS box to any desired size. The system font will be used on default.

1.4.2.6. Ability for a VxD or DOS app to change the title of a DOS VM

Priority	Status
2	Done

This feature will allow the name of the application to be displayed as the DOS VM title. A "ShellVMTitle" API is required to support this.

HIGHLY CONFIDENTIAL

1.4.2.7. Ability to close the DOS application gracefully

Priority	Status
2	Done

Currently Windows cannot be shutdown when a DOS VM is active. The ShutDown API will allow for the graceful shutdown of the DOS VM and hence Windows.

MX 2120386 CMS 00013680
CONFIDENTIAL

RBC 04081

1.4.2.8. COMMAND.COM enhancements

- A File "Search" engine will be used by DOS COMMANDS (e.g. DIR, TYPE, COPY etc.), when the user invokes them with special switches or syntax. The default behavior will the same as before to preserve compatibility. The newly implemented file search engine will support the following search criteria when invoked:

Priority	Status
2	Done

- multiple wild card specifications in a file name and extension (example: *a*b.*x*)
- multiple filenames as input parameters to DOS commands
- last modified date relative to literal date.
- last accessed date relative to literal date.
- file content-based searches
- file attributes-based searches (System, Hidden, Archive, Read-only)
- subdirectory traversal
- UNC path name support

- An option to increase DOS Command Line length and Environment Variable length

Priority	Status
2	Done

- Environment variable expansion on the command-line (example: SET PATH=%PATH%;C:\FOOBAR)

Priority	Status
2	Done

- DIR Command Enhancements

Priority	Status
2	Done

- DIR will display free disk space when disk is empty.
- DIR will display number of Files and Directories (instead of merging).
- DIR will optionally display last access date and file attributes

- COPY Command Enhancements

Priority	Status
3	Done

- COPY will prompt for new disk when target disk is full.
- COPY will prompt for confirmation if an ambiguous destination is specified, when the user specifies the /X (extended) option.
- XCOPY and REPLACE functionality will be merged into the COPY command. The commands XCOPY and REPLACE will also be shipped separately for reasons of compatibility with existing software.
- COPY will optionally use XMS/EMS memory for copy operations.
- REN command should also rename Directory (worry about 64 character limit)

Priority	Status
2·	Done

- COMMAND.COM should be smarter about reloading transient.

Priority	Status
1	Done

COMMAND.COM will be smart enough about reloading its transient portion. The following logic will be used to locate the transient:

- Search for COMMAND.COM in the location specified by the COMSPEC.
- If not found, Search for the COMMAND.COM where it was initially loaded from.
- If not found, Search for COMMAND.COM in the root directory of the boot drive.
- If not found, ask the user to specify the location of the COMMAND.COM and load it from the specified location. This prompt should be repeated until a proper COMMAND.COM can be located.

If at any step, a copy of COMMAND.COM is found that is from a different version of DOS (wrong checksum), the search should continue with the logic as if no COMMAND.COM was found.

- COMMAND.COM should load its resident data portion in UMBs.

Priority	Status
2	Done

COMMAND.COM currently loads its resident code into HMA but it still loads its data in the conventional memory. COMMAND.COM should be enhanced to take advantage of UMBs by default to load its data and resident stub code into the UMBs. This will give users another 3Kb-5Kb of free conventional memory on systems with UMB support.

- COMMAND.COM should use /E setting of resident COMMAND.COM

Priority	Status
3	

COMMAND.COM should inherit the /E setting of resident COMMAND.COM instead of adjusting the size of the environment to be the current environment. The current method of inheritance causes many problems with the batch files because batch files are not able to increase the environment. We could add a switch /O (old behavior) to COMMAND.COM to only inherit the current environment if a user needs more memory for his batch file.

1.4.2.9. New Commands

- START: ability to start a specified program in a new DOS VM.

Priority	Status
1	Done

- LOAD: ability to load device drivers from the command line.

Priority	Status
2	Done

- MOVE: ability to move files from the command line.

Priority	Status
2	Done

HIGHLY CONFIDENTIAL

The success of moving a sub-directory from one location to another, will be restricted by the 64 character limit that DOS imposes on the complete path name. In other words, the move will fail in the case where the end result of the move will cause the file at the leaf node to be more than 64 characters (when specified as a complete path from the root).

MX 2120388
CONFIDENTIAL

CMS 00013682

RBC 04083

1.4.2.10. Enhancement to Existing Commands

Priority	Status
2	Mostly Done

- DOSKEY:
- NO ECHO switch when executing DOSKEY Macro.
- An option to increase Keyboard Type Ahead buffer.
- Ability for DOSKEY to be able to read macro definitions from a file.
- Enable DOSKEY support in applications that use INT 21h/AH=0Ah (e.g., EDLIN, DEBUG, etc.).
- Disable "exit" from the first command interpreter when the user requests this option. This is intended as a means of preventing accidental exit from a Windows VM. An alternate command, such as "exit VM" will be provided so that the user can consciously exit the first command interpreter.
- Enable alternate history recall mode (i.e., no duplication of previous commands in the DOSKEY command stack, provided they have not been modified).
- Provide Screen Scroll-back feature. (Not Done)
- Integrate DOSKEY with COMMAND.COM to save low memory. (Not Done)

- Thousands separator for numerical output (MEM, CHKDSK, FORMAT, DIR)

Priority	Status
2	

[The need to provide an option to display numbers without the thousands separator, needs to be explored.]

- A paged output option for commands that generate more than one page output (TYPE, ATTRIB, MEM)

Priority	Status
2	

1.4.2.11. Support Extended Text Modes (arbitrary # of lines/columns)

Priority	Status
3	

There are lots of displays that support 132 column mode and more than 25 lines of text. We should provide a generic grabber that will support an arbitrary number of rows and columns. Compatibility issues, like DOS word processors that wrap based on specific column and row size, has to resolved.

1.4.2.12. Support CoolSwitch from a full screen DOS box

Priority	Status
3	

The CoolSwitch allows ALT-TAB cycling through the active applications from a DOS full screen, without launching the applications. Currently the full screen DOS box, just puts up a line. It would be nice to throw up a pop-up box.

1.4.2.13. Stream Oriented Cut and Paste

Priority	Status
3	

It would be nice to be able to cut a hunk of text and specify that it is actually a stream of text, i.e. CR/LF should not be added to the end of every line.

1.4.3. System configuration and DOS Application Configuration

1.4.3.1. Multiple configuration boot for easy system configuration

Priority	Status
1	Done

Multiple configuration boot allows the user to choose from one of many different configurations at boot time. This is implemented in ASTRO and will be provided in Chicago in the same format. Please refer to the Jaguar spec for more details.

1.4.3.2. Centralized Program Information File (PIF) Editor

Priority	Status
1	

Currently the configuration information for DOS and Windows applications is spread across multiple *.INI and *.PIF files. The intent is to consolidate this information into a central database. The implementation strategy for this has not been solidified at this point. The older files will have to be maintained for compatibility purposes. However programs can garner the configuration information necessary from a single database (or regisiry), as long as the information is kept up to date.

1.4.3.3. Local Configuration file

Priority	Status
1	

In Chicago, CONFIG.SYS and AUTOEXEC.BAT will contain global DOS VM configuration information. A per-VM configuration will be controlled by local configuration files. The actual implementation of this local configuration file has not been solidified at this point in time.

1.4.3.4. Automatically Run DOS Programs In Background Mode

Priority	Status
1	

Currently a DOS application running in a windowed DOS box, is suspended when it is in the background by default. Most users seem to be unable to figure out how to change this default option. In Chicago, the DOS application running in a windowed DOS box will not be suspended at all times when it is the background task. Whether the DOS box is suspended or allowed to run in the background will be based on information about what the application running in the DOS box is doing. For example, a COMM application will be allowed to run in the background, whereas a keyboard polling application will be suspended. This feature will allow a FORMAT or a DIR, for example, to continue to run when the DOS box is in the background.

1.4.3.5. Allow Access to the installed VxD List

Priority	Status
2	

The ability to traverse the installed VxD list will allow diagnostic software, like MSD, to provide information about what is installed in the system.

1.4.4. Size Issues

1.4.4.1. Fragmented, Shrinkable Swapfile

Priority	Status
1	

Currently Windows allocates contiguous physical disk space for its swap file. It might be particularly difficult to get a contiguous swap file on a system that has a fragmented file system. Further a fixed size swap file does not cater to the changing system and user needs. The fragmented, shrinkable swap file solves these problems.

1.4.4.2. An ability to Load VxDs Dynamically

Priority	Status
1	

Dynamically loadable VxDs are required to support Plug and Play. This feature will also configure the system to what is installed on the machine. Some VxDs will also be made pageable. [DEFINE WHAT VxDs].

1.4.4.3. Remove obsolete code used by previous versions

Priority	Status
2	

Since Chicago is tied to a version of DOS, older Windows code that dealt with different DOS versions is no longer needed. Similarly code in real mode DOS to support older Windows versions is no longer needed.

1.4.4.4. Increase number of pageable components

Priority	Status
2	

Various parts of the system, like DOSMGR, VxDs etc., will be made pageable.

1.4.4.5. Full INT 33h Mouse Support in a VxD

Priority	Status
2	

By moving the mouse support from a real mode device driver to a VxD will give us a zero footprint mouse support.

1.4.4.6. Use New Memory Manager Features

Priority	Status
2	

Various parts of the system, like Descriptor allocation, will be changed to use the newer set of services provided by the 32-bit protected mode memory manager. This will make the functioning of the system more efficient from a memory usage point of view.

1.4.4.7. Windows Kernel Should Release Memory for DOS DPMI Apps

Priority	Status
2	

On a system with low memory, a DPMI App might not be able to run because of its memory requirements. The Windows kernel can lend some of its memory to the DPMI app on a pageable system.

1.4.4.8. Support for 720Kb text mode DOS VMs

Priority	Status
3	

When a DOS application is running in character mode, the graphics memory in the segment range A000 to B000 could be used as conventional memory. There are several issues related to this feature that need to be resolved. For example, if the user runs a graphics mode apps, should we prevent the user from running this App or automatically switch to graphics mode. If we do switch to graphics mode, what hapens to information loaded in the A000 to B000 area. Such issues have not yet been resolved.

1.4.4.9. Move the hardware handling in Windows drivers

Priority	Status
3	

We should eliminate the duplication of code in a different parts of the system by moving the hardware handling in Windows drivers. This needs to be done for keyboard, mouse, timer drivers. This would lead to some performance improvements. One of the issues to consider is whether these new drivers and VxDs are supposed to work with old drivers and VxDs. For a cleaner implementations, the suggestion is not to make the drivers work with old VxDs, but to make the new VxDs work with old drivers. [We need to understand the implication of this suggestion].

1.4.5. Performance Issues

1.4.5.1. Provide shadow RAM for V86 ROM

Priority	Status
2	

As part of the low 1Mb memory scan, identify ROM areas and remap them with RAM for better performance.

1.4.5.2. Allow larger translation buffer in system VM than other VMs

Priority	Status
2	

The system VM supports all Windows apps. Hence having a large translation buffer, that translates p-mode addresses space to real mode address space, would enhance performance.

1.4.5.3. DOS Console Driver Should Cooperate With the VDD

Priority	Status
2	

Currently the DOS Console driver outputs a character at a time, causing a ring transition for every character for a windowed DOS box. Buffering this output to a per line basis, will reduce the number of ring transitions and enhance performance. This will be especially visible in commands such as DIR and TYPE.

1.4.5.4. Async Time-Out Notification

Priority	Status
2	

This feature will provide support for asynchronous time-out notification for VxDs. Such a service is required by disk device drivers to deal with pathological cases (such as a controller that went bad etc.)

1.4.5.5. General Performance Enhancements

Priority	Status
2	

System components such as VPICD, XMS driver, time-slice routine, primary scheduler etc. should be reviewed and optimized. Some system components like the Memory manager will be significantly enhanced and will support relevant Win 32 API services.

1.4.5.6. Dynamic cache memory allocation

Priority	Status
2	

Currently static memory is allocated at startup time and does not cater to the needs of the system. Dynamic allocation is more flexible and caters to changing user needs.

1.4.5.7. Reduce Critical Section Signals

Priority	Status
2	

Currently there are a lot of critical section calls to INT 2Ah. The intent is to avoid these critical sections if they are not necessary, defer them when possible if they are necessary and reduce the granularity so that only the part that actually needs the critical section is protected by the call. Further isolated critical sections that have no correlation between each other should not be locked in-step so that multiple code paths can execute at the same time. For example, DOS applications should be able to execute DOS calls 1-10h (for keyboard polling) when another VM is doing file I/O. All of these changes will enable smoother multi-tasking from the user perspective.

1.4.5.8. Happy-Time Events to allow VxDs to process an event in SYNC with VM

Priority	Status
2	

Ralph believes this would be very helpful for VxD writers. A happy time event is one that is executed when DOS is in a known good state. What he has in mind is that when Windows is switching from one App to another it would check to see if this queue has anything in it. If it does it would call to Ring 0 to process the events.

1.4.5.9. Avoid Idle-time processing when power management is active

Priority	Status
2	

During idle time, the system tries to do useful work, like mail services, disk defragging etc. However this kind of work should not be done when the system is under active power management.

1.4.5.10. Automatic Save/Restore of DOS and Windows State

Priority	Status
3	

This feature is important for several reasons:

- As we address the memory limitations in User and GDI, it will be possible to run lots of Windows applications concurrently. FastDisk already lets users run lots of DOS applications. The more applications you have to shut down at the end of the day, the more interesting this feature becomes.
- Laptop computers could turn themselves completely off, providing the ultimate in power-saving standby mode.
- This capability will provide the basis for support of super-exclusive mode. The state of the system would be saved, we would boot into real mode DOS, and when the user exited from the real mode DOS box, the entire system state will be restored.

However this feature could be difficult to implement and could be fraught with problems. In the case where the state of the system has not changed and the machine is not on the network (like most portables), this will work and would be a nice feature that allows for fast powerup. Even if the state has not changed, and the machine is on the network, at the very least all network connections will have to work on "restore". However if the software state has changed (more apps installed say) or the hardware state has changed (devices installed or removed), the problem of restoring the system and catering to the changed state becomes immensely difficult.

HIGHLY
CONFIDENTIAL

MX 2120396
CONFIDENTIAL

CMS 00013690

RBC 04091

1.4.6. Robustness Issues

1.4.6.1. Better Cleanup on VM termination

Priority	Status
1	

When a VM terminates, the system should get rid of all VM resources in use. This is usually done by associating the resource with a owner. The system should be able to find all resources associated with the owner and release them. This also applies to DPMI and XMS memory that the VM might have used. Currently, for example, the DPMI memory is not released properly when the VM is terminated.

1.4.6.2. A Crash of the System VM Should Not Be Fatal

Priority	Status
2	

Currently a system VM crash will bring down the entire system. The possibility of gracefully recovering from this, if at all possible, should be explored. It would be better to confirm any system state change that is being contemplated at this point, if it can be done.

1.4.6.3. NMI Watchdog Timer to prevent ring 3 software from disabling interrupts

Priority	Status
2	

User applications can currently degrade all other processing in the system by disabling interrupts for long periods of time. Systems that have an NMI watchdog timer, will be able to detect this condition and take appropriate action.

1.4.6.4. User control for more DOS VM protection

Priority	Status
2	

If the user chooses to run a "protected" DOS VM, a substantial portion of the VM will be instanced. This will provide more robustness for the system as whole, but can sacrifice some compatibility. The default would be to not compromise on the compatibility of the VM.

1.4.6.5. "Must Complete" Mechanism

Priority	Status
2	

This is a mechanism to delay the termination or suspension of a VxD when the thread being terminated or suspended is in a critical piece of code. This will allow critical pieces of software, like drivers manipulating hardware, to leave the device or software in a known state. [EXPLAIN WHY THIS IS DIFFERENT FROM A CRITICAL SECTION].

1.4.6.6. Debugging support

HIGHLY CONFIDENTIAL

Priority	Status
3	

- When the system integrity is compromised, more details needs to be provided for the user or the developer to identify the cause of the problem.

CMS 00013691

MX 2120397
CONFIDENTIAL

RBC 04092

- A ToolHelp VxD will provide standard APIs for third party debuggers.
- The Global memory will be protected, where possible, to detect stray pointer problems in running code. This will help with the development of VxDs.
- Using a task gate for the Double Fault handler, will allow the system to catch pathological cases and deal with it gracefully. When code that handles a violation itself causes another violation, the double fault handler is called. At this point, the system should go to a known state to allow the user or developer to gather more information about the cause of the problem.

1.4.7. DOS APPLICATION COMPATIBILITY

1.4.7.1. Top 100 DOS Apps Support

Priority	Status
1	

In general applications (including games) need to be analyzed and supported better. Special importance should be given to the 40 of the top 100 apps that are known not to run properly in a DOS VM under Windows 3.1. These 40 applications were identified by a summer intern who ran the top 100 applications on Windows 3.1 and documented their behavior.

An example of the kind of work that needs to be done, would be to virtualize the CMOS timer better so that games run properly. Currently the Sound support in a DOS VM is dismal. This needs to be resolved in Chicago because multimedia support is becoming more and more important.

[AUGMENT LIST OF APPLICATIONS WITH LIST FROM MARKETING AND TESTING].

1.4.8. WIN32 Support

1.4.8.1. Thread Support

Priority	Status
1	

- Support thread events
- Support thread time-outs
- Termination and suspension events for threads, instead of VMs, need to be supported
- The Floating point context needs to be either associated with a VM or a thread. Further investigation is required to see if it makes sense to make it part of the thread context, because most threads do not execute floating point instructions. [What is NT doing? If we choose to do FP per process or per VM (because of size issues in saving FP state per thread), should we provide a programmatic way for an application to request a per thread FP state?]

1.4.8.2. Support Mutex

Priority	Status
1	

Currently the only synchronization object that is well suited to mutual exclusive access of a resource is the critical section. This is only one instance of the critical section. Mutexes would be synchronization objects that could be created dynamically and used to protect resources.

1.4.8.3. Win32 File API support in VFAT

Priority	Status
1	

We need to implement Win32 File API support in VFAT. Some of these WIN32 APIs might not have an analogous DOS service and hence may not be implementable in Chicago. Other APIs would require explicit support in various FSDs and in the IFS manager. [ENUMERATE THE SUPPORT].

1.4.9. New Hardware support

1.4.9.1. Support for booting from a Large (>20MB) Floppy

Priority	Status
1	

At boot time, IO.SYS limits the types of floppies supported to one of six known types. It would be necessary to support higher capacity floppies in Chicago. Currently OEMs are producing 40MB floppies that the user should be able to boot from.

1.4.9.2. Detecting > 64 MB memory on ISA systems

Priority	Status
2	

[WHAT IS NT DOING? WE SHOULD SYNC UP]
The IBM defined INT 15 function (Get Extended Memory size, Fn 88h) can only return a maximum of 64MB memory. Apparently, Compaq has defined another function (fn 8A) that does not have this limitation. We should support this new function. (AMI wants to implement it in all of their BIOS, if we will use it)?

1.4.9.3. Support for large disks in real/protect mode (>1024 cyl)

Priority	Status
2	

Disks with cylinders greater than 1024 need to be supported in Chicago. Support will be required both in real mode and protected mode. This will be supported through the INT 13H extensions (either through the BIOS ROM or the Pre-load mechanism mentioned below). Currently existing systems (with their existing partitions) can be upgraded without loss of data, while being able to access any extra disk space that was previously unused. It is important to state that the CMOS settings in these cases will not be changed and Cougar will work with the system in a manner that would allow the Real mode to work correctly too. The INT 13H extensions and the Pre-load scheme, along with the new partition types are described in detail in the Jaguar Spec.

1.4.9.4. Removable Media Support

Priority	Status
2	

Newer INT 13 functions for lock, unlock and eject have been defined. These should be used both by real mode and protect mode drivers to support removable media in drives that features electronic locking. In general cache programs might be affected by this. These programs will have to flush their cache when the user decides to eject the media.

1.4.10. Device Drivers

Dragon is the name of the Layered Device Driver Architecture used in Chicago. FastDisk was the driver architecture used in Windows 3.1. An overview of the Dragon architecture and the FastDisk architecture can be found in the Appendix. The interested reader will also find brief explanations about Volume Tracking, TSDs, etc in this Appendix. These terms are used in the context of the Dragon drivers in this document.

1.4.10.1. Compatibility with existing FastDisk drivers and clients

Priority	Status
1	

Dragon Device Drivers will have to be compatible with existing FastDisk drivers and FastDisk clients, such as:

- Ultrastore
- Future Domain

- Gibson research cache
- Dumbo printers
- HyperCache
- Others (need to produce a list)

[EXPLAIN WHAT EACH DOES].

1.4.10.2. Compatibility with VFD for Norton and Central point Backup

Priority	Status
1	

VFD is the Virtual Floppy Driver in Chicago. The floppy device is virtualized in Chicago. In this case, when real mode programs try to access the floppy device directly, the access is trapped and emulated (or executed). The VFD should be checked for compatibility with utilities such as Norton backup and Central Point that access the floppy directly.

1.4.10.3. ASPI / Future CAM real mode device driver compatibility

Priority	Status
1	

These are two most popular architecture for Real Mode SCSI plug and play drivers. We must support these drivers.

1.4.10.4. Develop Dragon test programs

Priority	Status
1	

We need to develop a Dragon test suite that we could run against third party dragon drivers to make sure that they pass our test criteria. We may also have to setup a test lab for these drivers to verify their reliability.

1.4.10.5. Finish Dragon drivers

Priority	Status
1	

We need to write the following layers for dragon drivers to support the existing popular hardware.
TSDs (Target System Drivers)

- Disk TSD (Disk, Floppy, Magneto-Optical)
- CDROM TSD
- Tape TSD
- Printer TSD

SCSI-izer

- Disk/Standard SCSI-izer
- CDROM SCSI-izer
- Floptical SCSI-izer
- Tape SCSI-izer
- Printer SCSI-izer

We may have to write few other CDROM/Tape/Printer SCSI-izer for different manufacturers.

Port Drivers

- ESDI/IDE/ST506 port driver
  - support standard IDE controller
  - support alternate IDE controller
  - support Compaq Alternate controller (extdsk on docking station)
- NEC port driver
- Fast disk port driver
- Real Mode port driver
- ABIOS port driver
- Compaq IDA (integrated Device Array) port driver
- Standard parallel port driver
- Standard Serial port driver

1.4.10.6. Support for alternate IDE Controller

Priority	Status
2	

We should provide support for alternate (second) IDE controller in our Dragon disk device drivers. We have already implemented this support in INT 13 extension for IDE controller.

1.4.10.7. Support for Compaq's alternate controller

Priority	Status
2	

We need to understand and support Compaq's alternate disk controller. This is required to support Compaq's docking stations.

1.4.10.8. Volume Tracking

Priority	Status
2	

HIGHLY CONFIDENTIAL

The functionality of volume tracking for removable media needs to be determined. [Is it sufficient to have the functionality of DOS 5 volume tracking or do we need a fancy volume tracking??]

CMS 00013697

RBC 04098

1.4.10.9. New communication driver for Better COMM performance

Priority	Status
2	

We should write a protect mode Communication driver (a COMM VxD) to support higher communication through-put under Chicago. We also need to decide if Dragon driver architecture is suitable for Communication drivers.

1.4.11. File System

1.4.11.1. MSCDEX enhancements

Priority	Status
1	

MSCDEX is the real mode CD-ROM file system. We need to enhance MSCDEX to support Photo-CDs and fix the existing MSCDEX bugs. Support for Photo-CDs could be very trivial because Photo-CDs are recorded in ISO-9660 format and CD-bridge driver masks the multi-session information from MSCDEX.

When possible, the protect mode CD-ROM file system will take over the functions of MSCDEX. However there will be cases when the protect mode CD-ROM file system cannot take over the functions of MSCDEX. Hence it will be necessary to ship the MSCDEX file system also. This is no different from the need to support the real mode FAT file system, despite the existence of the protect mode FAT file system.

1.4.11.2. Protect mode Character File System Driver (CFSD)

Priority	Status
1	

The CFSD will support character mode device drivers. It will essentially be a name resolve for the most part, routing requests to the appropriate driver.

1.4.11.3. Protect mode FAT file system (vFAT)

Priority	Status
1	

A protect mode FAT file system driver (vFAT) will be implemented. vFAT will provide better performance because of the 32-bit code path. It will also be multi-threaded, allowing multiple application to execute through the code. This will lead to smoother multi-tasking if vFAT is used.

1.4.11.4. Booting From PCMCIA card

Priority	Status
2	

We need to define the requirements for booting from PCMCIA media. The definition work is pretty much done but we need to push this through the PCMCIA tuple committee. In addition to the definition we also need to do the following to make it possible to boot from PCMCIA.

- Implement file system boot code for "boot tuple"
- Modify IO.SYS init code to not read system files from Media.
- Build CardDrv and MS-Flash into IO.SYS as default drivers
- Modify MemCard utility to place boot tuple on media and copy system files to boot partition.

1.4.11.5. Compression VxD

Priority	Status
1	

We need to either put ASTRO Compression in VFAT and Flash FS or develop a Compression VxD that can be used by all file systems. Using a Compression VxD that can be used by all file systems is the recommended implementation.

1.4.11.6. Undelete support in File System

Priority	Status
1	

Currently undelete is implemented as TSR/utility in real mode. There are some good reasons to move this to be a file system function. However there could be compatibility problems with existing utilities, such as the Norton utilities. This approach needs to be explored.

1.4.11.7. Exclusive File System Access for file recovery tools

Priority	Status
1	

File system programs, such as CHKDSK and defragger, require exclusive access to the file system. In a multi-threaded environment, all other threads in the system will have to be locked out in this case.

1.4.11.8. Protect mode CDROM File System

Priority	Status
2	

We need to develop a protect mode CDROM file system to move CDROM FS out from the low memory The better performance of the protect mode CDROM file system will be more important for multimedia software. [PORT NT CODE?]

1.4.11.9. Tape Support

Priority	Status
2	

We need to look at Tape format that NT has defined for the tape and somehow support it in the file system or Backup program. Do we need Tape FSD?

1.4.11.10. LAST Access Date Support

Priority	Status
2	

Last Access Date support will be useful for various functions, such as, archives, undelete, searches etc. This support needs to be added to both real mode and protect mode FAT file systems. This support should be turned off by default. In other words, the user should have the option of turning this feature on. [It might be desirable to leave this feature of for all removable media, like floppy]. [ENUMERATE COMPATIBILITY ISSUES].

1.4.11.11. Support for Flash File system in DOS utilities

Priority	Status
2	

The Flash file system will be a real mode only file system. Since this is implemented as a networked drive, all file system utilities will not function on this file system. Further, some of the utilities, like the defragger, will not be immediately relevant to the flash file system. Any additional utilities that are needed for maintaining flash file systems need to be identified and developed.

1.4.12. Miscellaneous

1.4.12.1. Upgrade XMS driver to XMS 3.0

Priority	Status
1	Done

The standalone HIMEM.SYS driver now supports XMS 3.0, but the internal Windows XMS driver is still at 2.0. The internal driver should be upgraded to support 3.0.

1.4.12.2. Ability for a VxD to communicate with Windows app

Priority	Status
1	

Currently there is no facility for the VxD to communicate with a Windows application. However a Windows application can communicate with a VxD. This service will facilitate bi-directional communication.

1.4.12.3. Add service to return limits of available EMS and XMS.

Priority	Status
2	Done

This is required for the new WinMem program.

1.4.12.4. Support Simulation of Physical Mouse and Keyboard

Priority	Status
1	

This work is for physically disabled persons. This is still under investigation. We need to work with GegLo on this.

1.4.12.5. Support a Subset of DPMI 1.0

Priority	Status
2	

The sparse memory allocation DPMI API is the subset to be implemented.

- Func 0401H - Get DPMI Capabilities
- Func 0504H - Allocate Linear Memory Block
- Func 0502H - Free Memory Block work with Function 0504H blocks
- Func 0505H - Resize Linear Memory Block
- Func 0506H - Get Page Attributes
- Func 0507H - Set Page Attributes

1.4.12.6. Complete Name based VxD calling convention

Priority	Status
2	

Currently VxD are identified by an ordinal, that is the module ID. This makes the maintenance of VxDs and additions to the functionality hard. A symbolic name resolution at run time will provide great flexibility in the handling of VxDs, when they are dynamically loaded.

1.4.13. Wish List

1.4.13.1. Long file name support

Priority	Status
2	

Long file names are perceived to be user friendly. There are multiple schools of thoughts on the implementation of long file names: enhancing the FAT file system, providing a separate file system that can handle long file names etc. There are also compatibility issues with each of these choices and there will be a ripple effect on utilities, applications that assume 8.3 dialog boxes etc. This issue needs to be explored further and resolved soon.

1.4.13.2. Wishes of the Network folks.

Priority	Status
2	

- The load/unload order for the VxDs should be LIFO order. The newer load/unload VxD scheme should take into account "dependency" between the VxDs.
- An ability to bundle multiple VxDs into a NET386 file (along the lines of a WIN386)
- DOS_CWAIT func should be allowed to wait on a VM. VMs should also pass error code back on exit.

1.4.13.3. Wishes of the Multimedia folks.

Priority	Status
2	

- Some Multimedia components will be bundled with SDK/DDK packages and the Chicago system.
- A generic 2-way communication from Apps to Drivers (say VxDs) is required. In the Source multimedia process, the driver needs to communicate with an App and in the Render multimedia process, the App needs to communicate with the driver. [DOES WIN-32 SPECIFY THIS ALREADY?]
- A fast QueryClock feature would help the multimedia performance immensely. This should apply to both system and peripheral clocks.
- Some peripheral boards, such as the MS sound board, might have a clock. It would necessary for the driver associated with this peripheral to export this clock, so that the Logical Clock multimedia object can be associated with this clock. The system should also provide a fast acess, if possible, to query this clock.
- If a timer/clock can be dedicated for the use of Multimedia it would be great. [THE COUNT DOWN TIMER IS USED FOR TIME OUT SERVICES, IS IT NOT??]

1.4.13.4. Ability to edit environment variables (PATH) from the command line.

Priority	Status
3	Done

SET will allow for user's to modify the existing environment variable on the command line by providing DOSKEY-type editing features.

1.4.13.5. Generic Text Mode Grabber to simplify support for OEM specific displays

CMS 00013703

RBC 04104

Priority	Status
3	

Currently OEMs have to write their own grabber to support a display that does not have the standard VGA modes. It would be better to simplify the grabber interface by writing a UniGrab. This would allow OEMs to write a simpler mini-grabber to support their specific hardware. If one does not provide a mini-grabber, then only the standard text and VGA graphics mode will be available.

1.4.13.6. Super-tiny font in iconized DOS boxes

Priority	Status
3	

When a DOS box is minimized, make the icon contain a tiny version of the screen. This lets the user monitor a compile or a download that is taking place in the background. The tiny font could be used for this purpose. Alternatively a small dot and a bigger dot can replace lower case and upper case letters respectively.

1.4.13.7. COMMAND.COM Memory saving

Priority	Status
3	

COMMAND.COM has a resident data of roughly 3K for each copy out of which more than 1K is resident messages. The first command's resident messages could be reused by succeeding COMMAND.COMs resulting in more than 1K of savings per VM.

1.4.13.8. i586 Support

Priority	Status
3	

- Utilize the virtual interrupt flag on the i586 for performance
- Reflect interrupts directly into Virtual 8086 (DOS VM). This will save a ring transition (performance).

1.4.13.9. Multiple partition support on removable media in real mode

Priority	Status
3	

Currently DOS does not support partitions on removeable media. With the advent of drives like the Bernoulli drive, Flash cards etc., removeable media with large amounts of space are becoming prevalent. It is important for us to also be consistent between the view of drives between real mode and protect mode. Either partitions should be supported on both real and protect mode or it should not be supported on both. The recommendation is to support partitions on both, though floppies should still be able to run without partitions. [IS THIS POSSIBLE? WILL IT CAUSE TOO MUCH USER CONFUSION??]

1.4.13.10. Ability to pre-load BIOS extensions (hook INTs before DOS is loaded)

Priority	Status
3	

HIGHLY CONFIDENTIAL

MX 2120410 CONFIDENTIAL

CMS 00013704

RBC 04105

The pre-load mechanism defined in Jaguar spec can be used to pre-load BIOS extensions. There are several other benefits of preloading BIOS modules e.g. Fixing bugs in the BIOS, Preloading INT 13 extension driver to be able to boot from large drives and pre-loading compression driver to be able to boot from a compressed drive (refer to JagSpec.Doc for more details on this). Naveen Jain, as a Microsoft employee, has applied for a patent on the pre-load mechanism.

1.4.13.11. Find-Multiple API for higher performance

Priority	Status
3	

A FindMultiple API needs to be defined. It would return a user specified number of files, instead returning a file at a time. It could also return Last Access Date in addition to other directory information. [IMPACT OF LONG FILE NAMES?]

1.4.13.12. Booting from CDROM

Priority	Status
3	

We should think about specifying a method to boot from CDROMs. As operating systems get bigger and take up a lot of disks, it may be cheaper for MS to start delivering OS on CDROMs.

1.4.13.13. DESQview/Topview ScreenUpdate API

Priority	Status
3	

The ScreenUpdate API is apparently used by some apps to inform the OS that the video memory has changed. By taking advantage of this API, the display can be updated pretty quickly in a windowed box This is a performance feature. It is unclear as to how many and which apps use this feature. The basic feature is simple to implement, one should get an app which uses it to test this.

1.4.13.14. Combine Smaller VxDs (e.g. Parity, sound and timer) to Reduce Overhead

Priority	Status
4	

1.4.13.15. Case mapping and code page confusion.

Priority	Status
4	

Since case mapping tables vary from country to country, users may have problems exchanging files between users with different country setups. IPG has suggested we use a single unified case mapping table for code page 850, to help alleviate these problems. This will potentially create compatibility problems for users who have existing files created with the existing case mapping tables.

1.4.14. Internationalization

1.4.14.1. Get/Set NLSINFO APIs for better international support

Current DOS NLS settings are based on a country code and do not have any knowledge of the language. They also have a preset definition of NLS setting for each country. SETNLS/GETNLS APIs are defined to remove both these restrictions. The following information can be set/retrieved using this Set/Get NLSinfo APIs. This API will be called by the Windows Control Panel to inform DOS about the change in NLS setting.

•	Language	Three letter code indicating language. e.g. "ENU" for US English
•	Country name	Three letter country name. eg. "USA", for USA
•	Keyboard	two letter Keyborad layout code. e.g US for USA
•	Measure	Measurement system (0 = metric, 1 = English)
•	Lzero	Boolean value set 1 (TRUE) if fractions should have a leading zero
•	Digits	Number of digits after decimal point in numbers
•	NegCurr	Integer value indicating negative currency format: 0 = ($0) 1 = -$0 2 = $-0 3 = $0- 4 = (0$) 5 = -0$ 6 = 0-$ 7 = 0$- 8 = -0 $ 9 = -$ 0 10 = 0 $-
•	TLZero	Boolean value, set 1 (TRUE) if time should have a leading zero
•	s1159	Symbol to put after time between midnight and noon (AM)
•	s2359	Symbol to put after time between noon and midnight (PM)
•	ShortDate	First introduced in Windows 3.1, this string contains a "date picture" of the short date format. sShortDate accepts the values M, MM, d, dd, yy, and yyyy. See sLongDate for information about these values and pictures.
•	LongDate	this string contains a "date picture" of the long date format. Values that are legal in the date picture are as follows:

Value	Item	Format
M	Month	1-12
MM	Month	01-12
MMM	Month	Jan-Dec
MMMM	Month	January-December
d	Day	1-31
dd	Day	01-31
ddd	Day	Mon-Sun
dddd	Day	Monday-Sunday
yy	Year	00-99
yyyy	Year	1900-2040

MX 2120412
CONFIDENTIAL

Other NLS data is retrieved via Get_Country info or Get_Extended_Country_info APIs.

CMS 00013706

HIGHLY CONFIDENTIAL

RBC 04107

1.4.14.2. Localization without any Re-Compilation.

MSDOS utilities should be capable of being localized without recompilation (for every language) For this to happen, all messages will reside outside of the executable code. Further these messages will be dynamically loadable at run time. Actually all utilities will be shipped with at least one default language message built into their executable (.EXE file). The messages will be stored in the EXE_HEADER and the size of the EXE_HEADER adjusted accordingly. Apart from this a central file with messages will usually exist, so that EXE files without messages can display information using this central file. When a file is copied on to a new floppy and run, the messages in the EXE_HEADER provide the ability to display a message, even though the central file might not exist.

1.4.14.3. SetNLS Utility to Improve on 'Ease of use' and 'Memory usage'

The current implementation of changing code page, country settings and keyboard layout is very difficult to use and takes up a lot of memory. Some of the limitations of the current implementation is:

- Cannot change the country setting without re-booting the system.
- Country setting is pre-defined and cannot be easily customized by a user.
- No facility to change language of the system without complete re-installation of the system.
- To change a code page on the system the user has to do the following:
  Load DISPLAY.SYS and PRINTER.SYS in the CONFIG.SYS file
  Provide country settings in the CONFIG.SYS file.
  Load NLSFUNC TSR
  Use MODE PREPARE to prepare the code page and then use MODE SELECT to select the code page.
  Requires KEYB TSR to change the keyboard layout.
- Takes up a lot of memory in the system.
- Does not support code page changes on popular printers such as HP Laser jet.

Our goal is to remove the above limitations of the current implementation. Based on our goal the important objectives for the proposed implementations are as follows:

- Maintain 100% compatibility with the NLS support found in DOS 5.0.
- Provide a new mechanism that allows the user to alter NLS settings with a single command.
- Design a system with a small footprint so as to take up very little system memory.
- Provide code page support for EGA compatible screen devices and printers that accept download-able fonts in HP format.

Compatibility will be maintained by leaving the existing NLS related files (LCD.CPI, EGA.CPI, COUNTRY.SYS, KEYBOARD.SYS, MODE.COM, NLSFUNC.EXE, DISPLAY.SYS, PRINTER.SYS, and KEYB.COM) unchanged. The existing "Country Info" API (INT 21h function 65h) in the DOS kernel will also remain intact and functionally unchanged.
A new utility SETNLS will be written for changing system NLS information (country settings, language and keyboard mapping) and installing code pages on system peripherals (video and printers).

Since DOS currently does not keep track of all the NLS information required for dynamic system configuration, the proposed implementation requires DOS to maintain a new DOS 'NLS INFO' structure to keep track of system NLS information. This structure will be similar to the NLS information currently tracked by Windows 3.1. The information in the NLS structure will be accessed only through system APIs. DOS will provide GetNLSInfo API to provide information about the current configuration. A new API SetNLSInfo will also be added to support dynamic modification of the system configuration.

The SetNLS utility will call the Windows Control Panel, when executed from with a DOS VM. The SetNLS utility will fail, when called from Real Mode DOS. In other words, there will be no new way for the user to change the NLS setting outside of the Windows environment. The older complicated method of changing the NLS setting will continue to work from Readl Mode.

1.4.14.4. Multilingual installation

Now that all of our utilities, System kernel and Command.com are language independent (use dynamic message loading), we need a mechanism to set a system for a specific language. Multi-lingual installation, which is expected to be a part of the SETUP program, will configure a system for the user specified country and language.

SETUP program will use MESSAGES.SYS (file containing messages for all the languages that a system is being targeted for) and SETNLS utility to configure the system for the target country and language.

We will ship a set of tools with MSDOS OAK for OEMs to add support for languages that are not supported by us (Microsoft should provide OEMs with a small set of translated messages for some of the popular Latin languages).

MSG2MB (message to message binary) tool converts message for a specific language into a message binary format. The message binary format is described in detail in the earlier section (this is the format that eventually gets copied to the EXE header). OEMs needs to run this tool for each language that is not supported by MS.

MB2EXE (Message binary to Executable) tool takes the Message binary and adds them to the executable binaries. This tools adds the messages in the EXE header for .EXE files and catenates messages to the end of COM files. MB2EXE can either add a set of language messages to the executable files, replace an existing set of language messages or append to existing set of language messages. Obviously, an .EXE and .COM files can have several language messages built into it its header.

MB2SYS (message binary to Messages.sys) tool takes a set of message binaries for different languages and appends them in a MESSAGES.SYS file. This file is used by all the utilities to load the messages for a language that are not part of its executable (not stored in the utilities header).

OEM SETUP program will ask a user for the language and the country that a system is being configured for. Obviously the prompt will be in a default language chosen by each OEM (most likely English). Once Setup has the country and the language information, it needs to do two things:

- Verify if the selected language messages are already built into the utilities header, if not, then it will parse MESSAGES.SYS for the Message binaries for the specified language and use MB2EXE tool to add these messages to the user executables. User can ask setup program to create executables that have more than one specified languages in all the executables.
- Use SETNLS utility to configure the system for the specified country. Default (quick) SETUP installation will choose the proper defaults (code page, country tuples, keyboards etc) for the specified country. Advanced option in the SETUP program will let a user modify any of the default settings. User can change country default settings at any time after the system has been configured (SETNLS is defined in detail in the earlier section). Upgrade SETUP program will parse CONFIG.SYS to search for DISPLAY.SYS, COUNTRY = , PRINTER.SYS and REM these lines in the config.sys. SETUP program will also parse AUTOEXEC.BAT for NLSFUNC, MODE CP PREPARE, MODE CP SELECT and CHCP and REM these lines.

Upgrade SETUP program will use the settings for SETNLS based on the information it collected from parsing CONFIG.SYS and AUTOEXEC.BAT files. A user will, obviously, have a chance to change any of these settings. The programs NLSFUNC, DISPLAY.SYS, PRINTER.SYS and CHCP programs. should be made stubs that will prompt a user to use SETNLS for changing the country settings.

1.4.15. Other Issues

1.4.15.1. PCMCIA & Flash File System

Personal Computer Memory Controller International Association (PCMCIA) is a group that is interested in defining a bus architecture for memory cards on portable systems for the most part. Companies like Intel and Fujitsu, provide interface controllers that allow PCMCIA cards to be hosted on their controllers, which in turn plug into other bus architectures, such as ISA. PCMCIA defines a 68 pin connector that has 28 Address lines and 16 data lines. PCMCIA cards, such as flash memory cards, can be hot plugged into the PCMCIA bus.

Currently there is a real mode flash file system and a real mode PCMCIA driver. The flash file system is an extant based file system with garbage collection. The extant based file system allows allocation at a byte level. The current FAT based file system cannot be efficiently used on flash memory, because of the curious limitations of flash memory (which are not enumerated here).

The intent is to leave the flash file system as a real mode driver and to convert the PCMCIA driver into a Dragon protected mode driver. The Dragon architecture provides the basis for supporting the PCMCIA driver, such as asynchronous event notification. Since the cards can be hot plugged into a system, the memory can magically appear in the system at any time. The drivers need to be capable of handling the sudden appearance and disappearance of memory (much like a removable floppy medium). There is also enough information on the card about the type of device that it is and the cards are usually software configurable. Hence this needs to be integrated with Plug and Play.

Currently there is no work item specifically assigned to PCMCIA (to the best of our knowledge). The ownership of this issue (Chicago Vs Mobile Windows) and the resolution of the work item and schedule (if it is Chicago) needs to be taken care of. Bill Krueger is going to write a memo on his proposal of what we should do. Russ Arun will then take this proposal and will bring the ownership issue to a resolution.

1.4.15.2. Printing

The printing work in Cougar consists of the print subsystem that will reside at Ring 0. The print subsystem, will support the NT API from a Ring 3 application's point of view. The Windows GDI will be changed to use this newer API. The print subsystem will be implemented as a VxD. The architecture of the print subsystem is described in a separate document, written by Aaron Ogus.

Support for bi-directional printers needs to be solidified. This is an open item and Lin Shaw has the action of writing a memo on it at an appropriate time. The printing spec itself is owned by Eric Bidstrup.

1.4.15.3. Networking

The networking work is being done entirely outside of the core Chicago team. The work is managed by John Ludwig. There are 3 protocol suites of interest in networking:

- NETBEUI: a protected mode VxD for NETBEUI has been completely for the most part.
- Novell IPX/SPX: a real mode IPX/SPX stack will definitely exist. A protected mode IPX/SPX stack is expected to be available too, though its timing might be at risk for the first Beta Release (3/93). It is expected to be available for subsequent Beta releases and will be bundled into the product. The story of the Novell clients for the various services (print, remote terminal etc.) needs to be understood.
- TCP/IP: a protected mode TCP/IP stack will be completely rewritten from scratch. It will hopefully be done by Feb. However the schedule is not solid at this time. TCP/IP is not expected to be bundled in with Chicago. The claim is that the stack is not easy to setup and can be sold separately for those who wish interconnectivity at this level. It will be good to understand this decision, because bundling it in has its advantages

from a good networking support story point of view. Further, NT is bundling TCP/IP in (I think), and it will be more consistent to do the same for Chicago.

The Networking group would like to have some of the problems in Chicago resolved. Please refer to the subsection "Wishes from the Network folks" for a list of these items.

1.4.15.4. Multimedia

The Multimedia group would like to implement the Clockwork framework on all Window systems, starting with Chicago. The Clockwork framework defines various processes as OLE objects. Of special interest are three objects:

- Source filter: which is a producer of multimedia data. It usually interacts with a capture device, through a device driver, and output multimedia data. The data could be audio, video or any other multimedia datatype.
- Transform filter: is a consumer and producer of multimedia data.
- Render filter: is a consumer of multimedia data. It usually interacts with a store or a playback device, to store, display or play the multimedia data.

A Logical Clock manages the synchronization between the various data types. The Logical Clock can either be bound to the system clock or to a clock on a peripheral device. Various issues with synchronization of data types, such as clock skew, will be handled by the Multimedia group. However a fast query of the state of the clock is very important, because it is an oft repeated function.

There are various issues with regard to Multimedia that need to be handled by either the Chicago team or the Multimedia team:

- Currently Chicago has no plans to support any other file system other than the FAT file system. If the persistent data storage format defined by OLE 2.0, ISTORE, is dependent on the underlying file system (such as OFS), it might not be possible to implement this on Chicago. The current assumption is that ISTORE is just a file format and is not dependent on the underlying file system. [WE NEED CHECK ON THIS].
- Multimedia cannot do better than the "interrupt latency" of the system and the "resolution of the clock" that it uses. In Chicago, the real mode part of the system will be capable of turning off interrupts for long periods of time. Though attempts will be made to ameliorate this where possible (with facilities such as NMI watchdog timers), there could be periods where the system is basically frozen from the multimedia point of view. The Clockwork multimedia framework should be capable of dealing with these pathological cases (where an inordinate amount of time can elapse).
- The issue of MS sound board support and any other Windows Sound related issues needs to be closed.

1.5. Appendix

The Appendix provides overviews of the architecture of some system components. It is meant to provide more information about these system components that are referred to in this document. However this overview is by no means a replacement for the document that specifies these system components. The actual specification would be the place the interested reader should go to get all the information about the system component of interest.

1.5.1. Dragon Architecture - Brief Overview

1.5.2. FastDisk Architecture - Brief Overview

2. Configuration Manager

2.1. Introduction

This document discusses the details of how the current Chicago setup design makes use of a piece of code we have named the Configuration Manager. The Configuration Manager helps Chicago Setup install (and uninstall) system components once the base system is up and running. However, this ability is useful in the general case, as this discussion will point out.

There are probably a number of functions or features something like the Configuration Manager should have to help make Chicago a more plug and play system. We hope this discussion serves to not only clarify the Setup spec, but to also stimulate thinking about how "plug and play" configuration should be implemented in general.

2.2. What is a Configuration Manager

The Setup group views the Configuration Manager as a system resource which abstracts much of the work involved in setting up up or re-configuring a Chicago system. This system resource would be used by the initial Chicago Setup, after which it would be available for any application level or plug and play level Setup or maintenance utility.

The Configuration Manager is the system resource which insures that an application or device can be installed and configured so that it does not conflict with other applications or devices already in the system.
The Configuration Manager would also be responsible for maintaining the least common denominator requirements of each application or device, a very important requirement if there is a need to un-install an application or device regardless of the order in which it was originally installed.

The Configuration Manager is NOT a generic setup or maintenance program in itself. It is a resource which allows many different types of setup or maintenance programs to be implemented without each one having to individually handle the multitude of details involved in installing or un-installing an application or piece of hardware on a Chicago system.

The Configuration Manager is NOT by design a means to replace the current system of configuration files such as .INI and CONFIG SYS and AUTOEXEC.BAT files, but it would have the means to abstract the management of these files.

2.3. Purpose of the Configuration Manager in Chicago

The primary feature of Chicago is Plug and Play. We feel an important part of Plug and Play is being able to easily install and uninstall components in the Chicago system. Easy in a UI sense, but also easy in a programatic sense. To accomplish this, there needs to be a single method or paradigm for installing a component and for uninstalling a component. This is the only way to maintain control and ensure installing and uninstalling really is simple.

The Configuration Manager plays a key role in this. The Configuration Manager is a piece of code, a black box, which works with install and uninstall programs to setup or remove components in the system. Install and uninstall programs contain all the logic and simply tell the Configuration Manager what to do. This is partly done through the already defined Win32 Registry API.

In the case of installing a Chicago system, the Setup program first installs the base Chicago components, mostly kernel components. The uninstall functionality for these base Chicago components is in the Setup program, itself. However, once the base Chicago system is installed, the Setup program starts the Configuration Manager and uses the standard install procedure defined for Chicago with the Configuration Manager to install all other components. To uninstall all these same components, the setup program would use the Configuration Manager and the standard uninstall procedure, just as any other setup program would.

2.4. Basic Required Functionality of a Confirguration Manager

- Provide a robust and consistent method of installing new or updating old files.
- Provide a robust and consistent method of querying and updating the configuration information found on a Chicago system without having to worry about the minimum requirements of other applications or devices already installed in the system.
- Should be able to provide most of the functionality needed to un-install any application or system component.
- Provide seamless integration for the future Plug & Play specification.
- Provide a single Setup solution for applications and utilities which must work across all Windows platforms starting with Windows 3.0. This will allow the applications group and ISVs to write a single Setup which works across all of their target platforms.
- Ensure least common denominator requirements between all applications, drivers and hardware within the system.
- The ability to deal with the requirements of both real and protect mode Chicago environments.
- Be easily extendible to meet future Setup requirements.
- The Configuration Manager would need to handle cases where the user or old Setup programs alter system files themselves.

To satisfy the listed goals will require that a Configuration Manager be able to abstract most of the details of installing, configuring, updating or uninstalling any DOS or Windows application, DOS or Windows device driver/VxD, or a new hardware peripheral. The Configuration Manager should be able to handle similar operations in a generic manner to allow it to work across the needed spectrums, such as when doing file manipulations (copying, deleting, etc.), but there may be some specific needs for each group of operations which needs to be handled implicitly.

2.5. General Install Procedure

The setup group envisions using the Configuration Manager as a system service which a Setup program can make request to for querying the current system configuration and then based on this information make requests to copy or remove certain files in the system or to change the system configuration information. Upon completion of all of it's requests, the Setup program would signal it's completion of all transactions. The Configuration Manager would validate each request and perform the requested operations, while at the same time archiving any information or files needed for a later uninstall request. The Configuration Manager could either do the requests in real time or que the operations and perform them all at all at once, to allow more robust error checking and optimization the requested operations.

The following is a very general script for how the setup program for a given printer driver would work with the Configuration Manager to install a new printer driver in the system. Imagine this happenning as a result of a user inserting a floppy disk with a new printer driver in the system, having the printer driver appear on the Cairo desktop as an icon, and then having the user drag the icon to the Control Pannel or some similar icon/agent, indicating they would like to install this printer driver.

As will be discussed in the requirements section, this scheme demands several things. First, each component (object?) which can be installed or uninstalled needs a corresponding setup/install and uninstall program. In addition, install procedures involving the Configuration Manager are transaction oriented so that the Configuration Manager can store the transaction log for uninstall purposes. Naturally, care should be taken to anticipate the uninstall requirements and choose the granularity of the transactions accordingly.

CMS 00013713

In the example below, there is only one transaction, for clarity.

1. SETUP - Notifies CM wants to begin install procedure

2. CM - Returns a transaction number which Setup can reference for uninstall purposes at a later date. (Just how Setup stores this number is an open issue.)

3. SETUP - Query CM for existing printer info (drivers, ports, etc.)

4. CM - Returns query information (including current use info).

5. SETUP - processes information and determines if can proceed.

6. SETUP - UI for all selections, ie., what port, what printer model, printer setup, etc.

7. SETUP - Request CM to remove specific files from system if these will conflict with new files to be installed.

8. CM - Ques the requests or deletes requested files or decrements their use count. (archives files for uninstall, updates transaction log)

9. SETUP - Requests CM to copy list of files to specified directory.

10. CM - Ques the request or copies requested files (archives any files being overwritten, updates transaction log).

11. SETUP - Requests CM to delete configuration information relating to files removed in step 6.

12. CM - Ques the request or updates specified configuration info (archives log of changes, updates transaction log).

13. SETUP - Requests CM to create new configuration entries based on new files and selections made by user in step 6.

14. CM - Ques the requests or creates new information entries relating to files copied in step 8. (archives log of changes, updates transaction log)

15. SETUP - Notifies CM transaction is completed.

16. CM - Peforms any requests in the que, closes the transaction log and return OK.

2.6. Uninstall Procedure

For the uninstall procedure, imagine the user has selected an icon representing a printer driver and dragged this to the trash can.

1. UNINSTALL - Notifies CM would like to begin an uninstall procedure

2. CM - OK

3. UNINSTALL - UI logic to determine what components, if icon represents more than one, to uninstall.

4. UNINSTALL - Query CM for current information about components to uninstall.

5. CM - Returns query information.

6. UNINSTALL - Follows logic for how to uninstall each of the target components. In this case, only one component. Passes transaction number for the installation of this component along with request to uninstall the transaction to the CM.

7. CM - Restores files and configuration information to original state by following the transaction log.

8. UNINSTALL - Ad hoc additional steps as necessary. None in this case.

This is a very simple example. To make this procedure more robust, an uninstall transaction log and uninstall archive logs could be kept until the operation is completed. This way, if the power fails in the middle of a complex uninstall program, there is a standard and safe way to either continue or back out.

2.7. Configuration information management requirements

2.7.1. Create and manage global information.

- Hardware resource allocations such as machine type, hardware installed list, I/O port allocations, IRQ allocations, ROM space allocations, SCSI information (units numbers in use etc.),
- Real mode driver information such as load order requirements, revisions numbers, load options, associations to hardware, associations between real and protect mode drivers.
- Protect mode driver information (same as real mode information).
- Global information about applications such as autoload list, shared file etc.
- Boot information in CONFIG.SYS and AUTOEXEC.BAT for real mode and similar information for protect mode.
- System level files lists and uninstall logs.

2.7.2. Create and manage application / device driver / hardware specific information.

- Information about type of application or driver, OEM information and revision.
- Application specific requirements such as hardware allocations (for drivers), environment settings, files or buffers = requirements, required device drivers, autoload requirements.
- File information such as file lists and the directories used.
- Application interaction information such as OLE and DDE, dependencies on other applications, information on compatibility with other app.,
- User specific options such as execution parameters, associations, group information, PIF information for DOS applications.
- Application level uninstall logs.

2.7.3. Create and manage system specific information.

- System .INI files, AUTOEXEC.BAT and CONFIG.SYS, program manager group files, configuration files required by Cairo.

Allow applications to create and access private information they may require for there individual needs.

2.7.4. Configuration manager file manipulation requirements.

- Archive files being replaced or slated for deletion.
- Copy new or replacement files from file list.
- Update shared file counts and verify correct shared file versions.
- Delete specified files (for uninstall or cleanup after an upgrade).

2.7.5. Configuration manager uninstall requirements.

- Delete any files added during Setup.
- Update shared file counts and determine correct state for shared files.
- Restore any files deleted during Setup.
- Restore all system level and application level configuration information.
- Ensure least common denominator requirements between all applications, drivers and hardware within the system.
- Delete any misc. temporary files created and used by the component being installed.

2.8. Registry Contents

The Configuration Manager will present the appearance of managing a registry of information, but the information being managed does not have to conform to any particular conventions. For example when installing an application on a Win 3.x system the registry will really be .INI files and the CONFIG.SYS and AUTOEXEC.BAT files, but when installing the same application on an NT system the registry may be a single registration data file.

2.9. Other Configuration Manager Issues

- System locking would be needed to avoid problems with other processes which may change system or application files which are in the process of being updated by the Configuration Manager.
- How intelligence the Configuration Manager should be is still an open issue. The general consensus is that it should make writing a Setup program so easy that everyone will want to use it, but it should not try to do so much that it becomes an unmanageable jumble of special cased code.
- What are the specific requirements placed on the Configuration by Manager Plug & Play?
- What are the specific requirements placed on the Confirguration Manager by Cairo?

3. Setup Programs for the Chicago Product

3.1. Introduction

This specification covers the general architecture, design, and features of the Chicago setup effort. The upgrade setup program, OEM packaged product program, and OEM post install program are included in this spec. OEM pre-install tools are not included and will be addressed in a later document.

Don't expect the feature set of setup to be short and simple because the setup program, itself, is not. Nor can it be. From the perspective of upgrading or setting up a system on a naked machine, Chicago is very different from Win 3.1 or MS-DOS 5.0 in that in covers both the DOS issues and the Windows issues. Issues which further complicate the Chicago setup design include:

- The code base Chicago is bigger than the combination of MS-DOS 5 and Win 3.1, and growing
- The MS-DOS 5 setup program and the Win 3.1 setup programs total over 550k, and setup certainly grown for MS-DOS 6 and Sparta products.
- The current *Getting Started* books for MS-DOS 5 and Win 3.1 are a total of 260 pages, not including the OS2 upgrade information added for Janus.
- Compatibility and upgrade issues have only gotten worse with all the new shell add-on products, hardware, and network products which have shipped since MS-DOS 5 and Win 3.1
- The major focus of the Chicago product is plug and play, which means, in part, making system setup and configuration even easier, simpler, and safer

The general goals and features for setup are covered first in this document. This section is important because it defines what setup is and what it is not, what setup will do and what it will not do. The next sections on architecture and design provide a framework for how the setup program pieces are constructed, where they fit into the program flow, and where various features will be implemented within each of the programs. The last sections cover open issues, outside dependances, and next steps.

There are several related documents which are important to review before or while reading this specification. The "Chicago Setup Project" document covers the various setup programs which we need to produce, the target audiences, and the high-level scenarios for each program. The "Plug and Play in the Chicago Product" covers what plug and play means and how the setup programs fit into this framework.

3.2. Goals for Chicago Setup

The general goals for the Chicago setup can be broken into two major categories: goals which relate to product features with end user benefits and goals which relate to how to produce the various setup programs in an efficient way. Both are important, many aspects are interrelated, and considerable program management and development time will be spent on each category.

3.2.1. Feature Goals

The two second summary:

- Upgrade DOS 3.1 or higher systems, may have Windows 3.X, also work on naked machines
- Always be able to return to the original system, complete uninstall capability
- Handle all the special cases which the MS-DOS and Windows setup programs covered, plus all the new cases uncovered during beta testing
- Work well over a network
- Provide batch install capability
- Make setup fun and entertaining

A summary list of features appears in the tables below. In this first draft, most of the goals and features relate to end user benefit, but some relate to design and architecture. Depending on what people find most useful, these will be separated in later drafts. Also, some of the features are a little fuzzy. For example, "Make setup flexible enough to accommodate the needs of corp MIS people setting up typical workstations in 1994." In the Appendix, these more fuzzy features have issues listed under them indicating that more information or a more crisp problem definition is needed. The thinking here was that it was better to have placeholders for areas we believe we will need to address rather than just omit them.

The tables are grouped into sections. Sections which apply to all three products appear first. Sections which are very specific, ie setup on an NT only system, appear last. A table of each feature with its relative priority and whether it is for only the upgrade (U), only the OEM packaged product (PP), only the OEM post install setup program (PI), etc. appears at the beginning of each section. The priority scale is: 1--important, will do; 2--will do opportunistically; 3--probably will not do.

List of Feature Sections:

1. General
2. User Interface
3. International
4. DOS only Upgrade
5. Hardware Detection and Device Drivers
6. DR DOS only Upgrade
7. Naked Machine Case for Upgrade
8. OS/2 Only Case for Upgrade
9. NT Only Case for Upgrade
10. DOS + Windows 3.X Upgrade
11. Chicago Upgrade
12. Upgrading a PC on a Network
13. Upgrade Over a Network
14. Upgrade Using a Portable
15. Batch Mode
16. Chicago + 1 Setup
17. Explicit Non-Goals for Chicago Setup

3.2.1.1. General: Feature/Goal

Feature Description	Product	Priority
1.1. Minimize floppy disk space by using various compression schemes	U, PP	1

1.2 Make the setup program small	U, PP, PI	2
1.3. Make setup fast	U, PP, PI	2
1.4 Make the setup program modular	U, PP, PI	1
1.6 Have setup run in DOS 3.2 and higher and in Chicago	U, PI	1
1.7 Warn user if there is a problem	U, PP, PI	1
1.8 Make setup fail-safe so that if power fails at any point, user can always recover to a workable system	U, PP, PI	1
1.9. Have uninstall of higher level components (ie, beyond Cougar/Chicago base) follow a general uninstall paradigm for Chicago using Configuration Manager/Registry API.	U, PP, PI	1
1.10. Uninstall completely, back to previous state except in a few OS2 upgrade cases	U	1
1.11. Option to store uninstall information on an INT13 local drive rather than a floppy	U	1
1.12. Keep log of setup progress so can easily recover. diagnostics for PSS.	U, PP, PI	1
1.13. Scan read/write disks for viruses	U, PP, PI	1
1.14. Look at destination disk and do a memory scan for viruses before installing Chicago	U, PP, PI	1
1.15. Provide option to create bootable "emergency" floppy (in addition to uninstall floppy).	U, PP, PI	1
1.16. Have bootable disk with diagnostic tools included.	U, PP, PI	1
1.17. Improve editing of autoexec and config.sys files	U, PP, PI	1
1.18. Allow user preferences to be 'sticky' when appropriate.	U, PP, PI	1
1.19. Capability to install Debug version or Retail version (for Betas, SDKs, Internal use).	U, PP, PI	1
1.20. If the user selects to upgrade over an existing installation, should delete all files associated with only the old installation (either DOS or Win) except 3rd party files like enhanced printer drivers	U	1
1.21. File copy control - control file attributes, file association (several files for one component), move file, delete file, delete associated files, what to do if file already exists, what to do if version conflict, different language, uninstall control, etc.	U, PP, PI	1
1.22. Clean up unneeded files on user's system after setup is completed	U, PI	1
1.23. Take advantage of multi-threading and muli-VM environment of Cougar to make program execution more efficient.	U, PP, PI	1
1.24 Option to checksum files on hard disk to verify file integrity when copying or moving.	U, PP, PI	2
1.25. Allow custom specification of bulk changes to Configuration files (AUTOEXEC.BAT, CONFIG.SYS, *.INI), unrelated to Setup files, to aid corporate admins in configuring machines	U, PP, PI	2
1.26. Automate creating a backup set of setup disks.	U, PP, PI	2
1.27 Have setup work off a CD ROM	U, PP, PI	2
1.28. Provide API to apps setup programs so they can take advantage of setup core engine functionality for file copy, etc.	U, PP, PI	2

1.29 Utility for Configuration editing (AUTOEXEC.BAT, CONFIG.SYS, *.INI): provide context sensitive help on every entry.	U, PP, PI	2
1.30 Capability to browse/print README files from Setup or have context sensitive help for exception conditions	U, PP, PI	2
1.31. Integrate Memory Optimization/UMB configuration with Setup	U, PP, PI	2
1.32 Support most popular software licensing schemes	U, PP, PI	1

3.2.1.2. User Interface: Feature/Goal

Feature	Product	Priority
2.1. Minimize keystrokes and mouse gestures by adopting a common and simple UI throughout setup	U, PP, PI	1
2.2. Make UI of setup consistent throughout setup, that is, if there are two modes, CUI and GUI, make them consistent so the user doesn't get confused	U, PP, PI	1
2.3. Use Wizards for hard tasks (such as finding a network log in script)	U, PP, PI	1
2.4. Make Setup fun and entertaining (whether with full UI or in interactive batch mode)	U, PP, PI	1
2.5. Make the setup program so the "engine" part of the program is independent from the UI.	U, PP, PI	1
2.6. Provide support for handicapped users in setup by allowing speech devices to be able to vocalize setup text	U, PP, PI	1
2.7. Enhance UI for floppy disk change requests	U, PP, PI	1
2.8. Option for people who set up for others but are not sophisticated enough to use a batch file: allow user to answer all questions, specify all choices up front and then have setup run until completion unattended unless a fatal error occurs.	U, PP, PI	2

3.2.1.3. International: Feature/Goal

Feature	Product	Priority
3.1. Detect code page, country code, and language settings at beginning of an upgrade and run setup using these settings.	U	1
3.2. Design all INI, INF, SSH and registry files so can be easily localized	U, PP, PI	1
3.3. Provide Sub-Language specification in setup and configuration. This should allow setup to ID counties in a unique way. NT uses the same method.	U, PP, PI	1
3.4. Design all setup components so they can be localized without recompiling the source code. Use the 'no-compile' multilingual design for Setup, as defined by the Jaguar spec.	U, PP, PI	1
3.5. Integrate the Base and Shell NLS specification and UI.	U, PP, PI	1

3.2.1.4. DOS only Upgrade: Feature/Goal

Feature	Product	Priority
4.1. Work on DOS 3.2 and higher	U	1
4.2. Provide a way to disable uninstall for those people who don't need this feature, mostly corp. accounts or resellers.	U	1
4.3. Detect DOS version and rename utilities to avoid name conflicts	U	1
4.4. Detect odd hard disks and disk drivers and provide simplified procedure for installing in these cases: Everex, Quantum, Plus Hard Card, and SyQuest	U	1

4.5. Detect drivers which offer >1024 cylinder support and convert these to the support scheme in Jaguar: Speedstor Bootall, Speedstor, Disk Manager, Praim, and Vfeature Deluxe	U	1
4.6. Detect situations in which drive letters changed in MS-DOS 5 and 6 setup and set up Chicago with same drive letters	U	1
4.7. Detect incompatible BIOS and warn user	U, PP, PI	1
4.8. Detect incompatible Hardware and warn user	U, PP, PI	1
4.9. Detect bad TSR's and warn user	U	1
4.10. Detect and fix up logical sectoring	U	1
4.11. Detect and fix up non-standard partition tables, numbers, such as the ones OEM's introduced when they modified older versions of MS-DOS	U	1
4.12. Option to detect incompatible utilities and applications and warn user	U	1
4.13. Detect and replace memory managers as needed	U	1
4.14. Keep current memory manager settings. IE, if 512k is specified for EMS and a certain memory range is excluded, keep these settings.	U	1
4.15. Update or replace disk cache. Detect presence of disk cache other than smartdrv.*. Ask the user if they want to replace with smartdrv.exe. Keep current settings, ie if they use 256k of memory for caching, keep this parameter.	U	1
4.16. Improve A20 line control with Himem. Get data from PSS.	U, PP, PI	1
4.17. Option to add compression to FAT file system (default). Detect existing compressed volumes and convert them to the Chicago flavor of Magic Drive.	U	1
4.18. Option to configure printers.	U, PP, PI	1
4.19. Option to run tutorial or autodemo(s) on Chicago	U, PP, PI	1
4.20. Add mouse support in part of setup before the GUI shell is launched	U, PP, PI	1
4.21. Allow user to specify different paths for core vs non-core components of Chicago to lower space requirements on physical hard disk used to boot	U, PP, PI	1
4.22. Selective install of optional components, similar to how Win 3.1 setup handles now in express and custom modes	U, PP, PI	1
4.23. Flex Boot feature for beta testers and general use. This would automatically be used if the user had enough disk space. This would setup up system so user could boot between their old system and their new Chicago system.	U	1
4.24. Option to scan hard disk for apps and automatically setup for Cairo Shell. Depending on the Chicago implementation of Cairo, there is a lot we can do here. Can generate Klasses, object attributes or properties, scan objects with special filters to build index files, etc.	U	1
4.25. Way to search disk for utilities and applications and set them up using Config manager and Registry API. This is not the same as for the Cairo shell. Here, information such as memory requirements, application uninstall information, and system configuration information would be included.	U	1
4.26. Need to handle working with ad hoc programs launched in autoexec.bat which do not return control to setup program or Chicago during a reboot.	U	1
4.27. Provide a better way for those who must backup and re-partition to install	U	1
4.28 Allow user to run a backup program before running setup	U	2
4.29. Do a better job of detecting and correctly naming DOS applications	U	2
4.30. Handle configuring MultiConfig MS-DOS 6 machine. Provide a mechanism to support booting using different configuration parameters/options in Chicago.	U	2

3.2.1.5. Hardware Detection and Device Drivers : Feature/Goal

Feature	Product	Priority

5.1 Detect SCSI devices	U, PP	1
5.2 Detect EISA devices	U, PP	1
5.3. Detect MCA devices	U, PP	1
5.4 Detect ISA devices	U, PP	1
5.5. Detect video controllers	U, PP	1
5.6. Detect IDE controllers	U, PP	1
5.7. Detect CPU/NDP	U, PP	1
5.8. Detect BIOS	U, PP	1
5.9. Detect the amount of RAM	U, PP	1
5.10 Detect I/O ports (COM, LPT, Game, etc)	U, PP	1
5.11. Detect other motherboard devices	U, PP	1
5.12. Detect sound cards	U, PP	1
5.13 Detect net cards	U, PP	1
5.14. Detect 3270 and 5250 emulation cards	U, PP	1
5.15 Detect and configure PCMCIA devices	U, PP	1
5.16. Detect device drivers loaded into memory	U, PP	1
5.17. Detect device drivers (real mode, old VxD's) which will not work with Chicago and warn user	U, PP	1
5.18. Provide a way to replace old device drivers (real mode, VxD's) with newer Dragon Drivers in a fail safe way	U, PP	1
5.19. Allow user to very easily upgrade a third party device driver	U, PP	1
5.20. Ability to disable/over-ride hardware detection during Setup and post-install.	U, PP	1
5.21 Use safe software methods to detect hardware as much as possible and use the result to fill in an exclusion map for I/O and memory. Detect the rest of the categories using riskier software or hardware methods only if they are essential to boot.	U, PP	1
5.22. Log detection progress so that in case of system hang and reboot, setup can intelligently continue at the stopping point and skip the dangerous detection that hung the system.	U, PP	1
5.23. Recognise whether a bi-di device is present on the serial or parallel ports	U, PP	1
5.24. Hardware support in transition to p-mode operation	U, PP	1
5.25 Optimize driver parameters	U, PP	1

3.2.1.6. DR DOS only Upgrade: Feature/Goal

Feature	Product	Priority
6.1. Work on DR DOS 5 and higher	U	1
6.2. Detect hard disk password protection and ask user to remove	U	1
6.3. Automatically remove delwatch utility and delete all undeleted files on hard disk	U	1
6.4. Delete all DR DOS specific utilities and files	U	1
6.5. Detect DR DOS version of SuperStore and offer user option to convert volume to Chicago compressed volume	U	1
6.6. Convert DR DOS/Netware Lite product to Chicago.	U	1
6.7 Handle DR DOS dual boot feature	U	2

3.2.1.7. Naked Machine Case for Upgrade: Feature/Goal

Feature	Product	Priority
7.1. Setup of a naked machine.	PP	1
7.2. Detect no partitions and give user ability to configure hard disk	PP	1
7.3. Detect partitions have not been formatted and give user ability to format them	U,PP	1

7.4 Give user option to add disk compression	U, PP, PI	1
7.5. Have uninstall automatically disabled in this case	PP	1
7.6 Allow complete NLS specification (Country, Keyboard, Language, etc.)	PP, PI	1

3.2.1.8. OS/2 Only Case for Upgrade: Feature/Goal

Feature	Product	Priority
8.1. No restriction on OS/2 version	U, PP	1
8.2. Offer to reformat FAT partitions to remove OS2	U, PP	1
8.3. Offer to repartition HD if non-FAT partitions detected	U, PP	1
8.4. Setup will not automatically preserve dual boot capability in systems with the old version of dual boot, will preserve dual boot with new version	U, PP	1
8.5. Preserve OS2 system settings	U, PP	2
8.6. Boot manager conversion. We should investigate feasibility of converting Boot manager to NT Flex Boot model.	U, PP	2
8.7. Retain OS/2 application settings.	U, PP	2

3.2.1.9. NT Only Case for Upgrade: Feature/Goal

Feature	Product	Priority
9.1. Detect NT and work correctly with Flex Boot feature	U	1
9.2. Preserve system settings (NLS information., network settings, shell configuration, etc.)	U	1
9.3. Replace NT drivers with Chicago versions, or add Chicago versions of NT drivers in Flexboot case.	U	1
9.4. Retain NT application settings	U	1
9.5. Utilize NT OFS (Cairo shell) information to generate OFS information for Chicago-Cairo shell.	U	1
9.6. Be able to read 3rd party setup.inf files in NT file format	U	2
9.7. Offer to repartition HD if non-FAT partitions detected.	U	1

3.2.1.10. DOS + Windows 3.X Upgrade: Feature/Goal

Feature	Product	Priority
10.1. Option to uninstall Chicago back to orignal DOS + Win	U	1
10.2. Uninstall works on DOS 3.2 or higher and Win 3.0 or higher	U	1
10.3. Update printer drivers	U	1
10.4. Update display drivers	U	1
10.5. Scan Win and Win/System directories and deal with any files which will cause problems later (ie. setup.exe in system directory for Win 3.1 setup)	U	1
10.6. Preserve startup and other groups as makes sense with Cairo shell	U	1
10.7. Scan hard disk for Windows apps, DOS apps and generate Cairo shell stuff (object klasses, object properties, index files, etc.)	U	1
10.8. Scan hard disk for Windows apps, DOS apps and generate Config Manager/Registry information	U	1
10.9. Support and/or convert Win 3.1 OLE Registry to new Config. Registry model.	U	1
10.10. Allow multiple versions of Windows to reside on the system.	U	1
10.11. Way to use all workstation/config/user specific information from existing DOS/Win installation to setup Chicago without destroying old config. Even if config exists on a separate machine.	U	1

10.12. Provide an option to also copy most popular unused drivers/components to the hard disk so that control panel utilities can reconfigure the system later without asking for floppies or net connection	U	2

3.2.1.11. Chicago Upgrade: Feature/Goal

Feature	Product	Priority
11.1. Ability to run under Chicago without going into super exclusive mode	U	1
11.2. Be smarter about copying files which have not changed to speed up setup.	U	1
11.3. Address file version checking issues	U	1
11.4. Ability to run setup over a modem if the async Winball stack is available from mobile group	U	2
11.5. Ability to uninstall components which were previously installed	U	2

3.2.1.12. Upgrading a PC on a Network: Feature/Goal

Feature	Product	Priority
12.1. Detect network card and configuration	U	1
12.2. Detect network software and configuration	U	1
12.3. Maintain network login scripts and the like	U	1
12.4 Maintain network volume redirections	U	1
12.5. Maintain network printer connections	U	1
12.6. Maintain network connections for client/server applications	U	1
12.7. Must be able to restore the original network software configuration as part of uninstall.	U	1
12.8. Allow saving uninstall information to a net share as an option	U	1
12.9. Upgrade network software during Setup.	U	1
12.10. Must be able to upgrade the network even when installing off diskettes and the network has not been started.	U	1
12.11. Separate components into shared (on the network) and non-shared (local to the PC) components. (Equivalent of Win 3.1 SETUP /A & /N).	U	1

3.2.1.13. Upgrade Over a Network: Feature/Goal

Feature	Product	Priority
13.1. Provide equivalent net log in and all original connections after setup has completed	U	1
13.2. Be able to begin setup with just a few floppies and then finish setup over a supported network	U	1
13.4. Remote setup and configuration of a workstation from a server.	U	1
13.5. Have setup work with popular software management schemes (Hermes)	U	1
13.3. Be able to begin setup with just a few floppies and then finish setup over a modem	U	2

3.2.1.14. Upgrade Using a Portable: Feature/Goal

Feature	Product	Priority
14.1 Ability to upgrade from an InterLink drive/connection	U	1
14.2. Ability to upgrade from a LapLink, Brooklyn Bridge, etc. drive/connection	U	1

MX 2120430
CONFIDENTIAL

CMS 00013724

RBC 04125

14.3. Ability to use a portable to backup, repartition, reformat local hard disk, and then install Chicago and restore files	U	1

3.2.1.15. Batch Mode : Feature/Goal

Feature	Product	Priority
15.1. Two staged setup to create batch file.	U	1
15.2. Ability to easily edit batch control file	U	1
15.3. Features to facilitate debugging	U	1
15.4. Flexibility in batch control to allow easy integration with corporate upgrade procedures and policies	U	1
15.5. Default batch mode which will "pop out" of batch mode if an error is encountered. All choices are default choices. Perhaps this is just express setup. No questions after user name and company registration.	U	1
15.6. Silent install - allow batch to specify error logging file with no interaction other than halting for fatal error	U	1
15.7. Ability to upgrade without any UI, separate task in a background VM, changes occur when user reboots	U	2
15.8. Ability to remotely upgrade a DOS workstation (Hermes?)	U	2
15.9. Ability to remotely upgrade a Win 3.X workstation (Hermes?)	U	2

3.2.1.16. Chicago + 1 Setup: Feature/Goal

Feature	Product	Priority
16.1. Make Chicago components, setup, config such that very easy to setup next OS after Chicago with ability to uninstall, retain net connections, etc.	U	1

3.2.1.17. Explicit Non-Features/Goals for Chicago

17.1. We will not enhance the character mode FDisk utility (need to review this as Davicol has recently questioned this)

17.2. We will not be able to install two different shells at the same time

17.3. Option to run disk defragger during post install phase
- If the user does this, we can not uninstall. Thus, should probably offer as a control panel feature rather than a setup feature.

17.4. Support of dual boot option for DR DOS systems

17.5. Run in a Win 3.0/3.1 DOS box.
- We will investigate if we can automatically shut down Windows and run under DOS after getting user's permission.

17.6. Provide way to combined FAT partition types 1 and/or 4 into a type 6 automatically
- Marketing calls this the 32 and 8 problem, referring to the situation in which DOS 3.3 is being used on a 40 MB hard disk.

3.2.2. Project Goals

The setup programs were critical path in the MS-DOS 5 and the Win 3.1 projects for a number of reasons. We want to avoid becoming critical path because the setup program is inflexible, difficult to maintain, or just generally buggy. To achieve this, we require a setup architecture which allows us to do the following:

- Make changes throughout the project as easy and painless as possible
  - Separate UI code so shell changes have minimal impact.
  - Allow new functionality to be easily added
  - Isolate the impact of fixing a bug
  - Make it easier to find and isolate bugs
- Have other groups supply install modules for their pieces
  - Ease of install is designed in from the beginning
  - Setup team doesn't have to be experts on every piece of Chicago (this was a problem in Win 3.1)
- Re-use code whenever possible
  - Re-use existing code from Janus and WFW setup
  - Use a dll design developed for Jaguar so that dll's can be used in both real mode and in protect mode
  - Re-use code for OEM pre and post install tools/setup programs

In addition, we will ensure the detailed design specs are always clear and complete[1], maintain strict code check-in rules for developers, and work closely with testing and the beta group to ensure the setup code is being efficiently and completely tested.

[1]A major point of the MS-DOS 5 and Win 3.1 post mortum reports

3.3. Software Architecture

This section covers the proposed architecture for the Chicago setup programs. This general architecture will provide a framework for thinking about how and where product features should be implemented in each of the programs. The next major section uses this framework to describe the basic program flow in the upgrade, OEM packaged product, and OEM post install cases. Note that the general architecture proposed here allows much of the same code to be used in all three of these cases, so in many future references to "setup", no distinction is necessary.

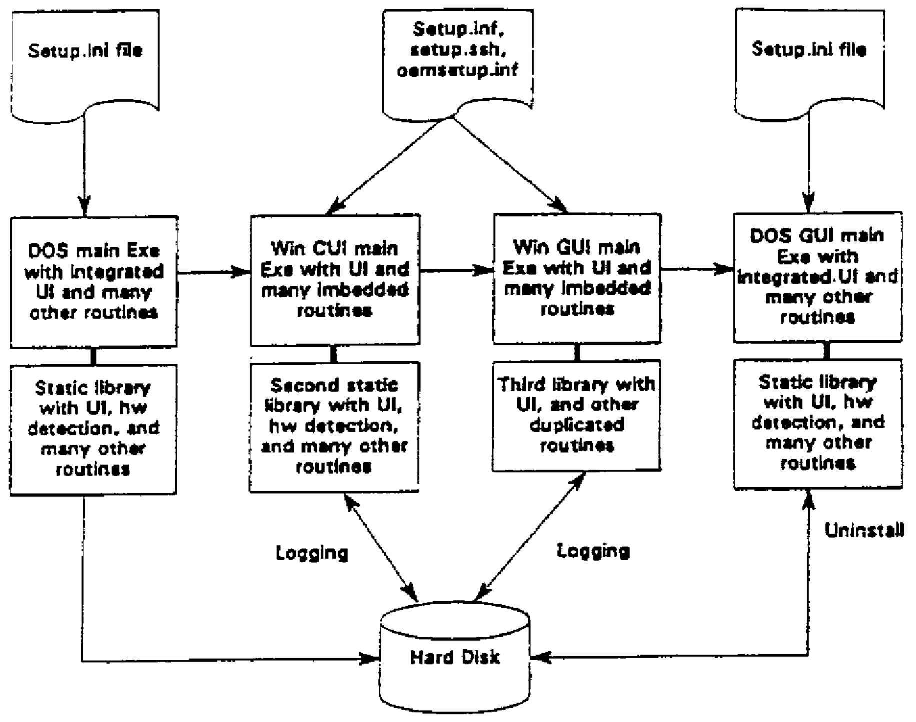
3.3.1. The Janus Approach

A lot of effort has been invested in the MS-DOS, Windows, and Janus setup programs. The architecture and design approach proposed for the Chicago setup programs borrows heavily from these earlier projects. We have the explicit goal of re-using as much of the existing code as possible, and we anticipate reusing over 60% of the Janus setup code.

However, we just can't take the Janus approach of gluing existing setup programs together and satisfy our project goals. First, the MS-DOS and Windows exe programs are very monolithic. The UI functionality and many basic routines are embedded in the main programs. There are many complex interdependencies between many of these routines. Many of these routines are not robust and have been tweaked during the beta cycles to work only in specific cases. Second, the Windows setup program was not designed with the uninstall feature. It is not possible to efficiently add this feature in a robust way using the current architecture of the program. Finally, there are a number of cases in which basic functionality is duplicated between the various modules. However, the code is not the same, so the exact functionality differs. This makes it that much harder to provide consistent services to modules we will want to use from other groups, or add ourselves, as the project goes on.

A diagram of the general architecture of the Janus setup programs is shown below[2]:

[2]The MS-DOS only and the Win only install cases are not shown for simplicity

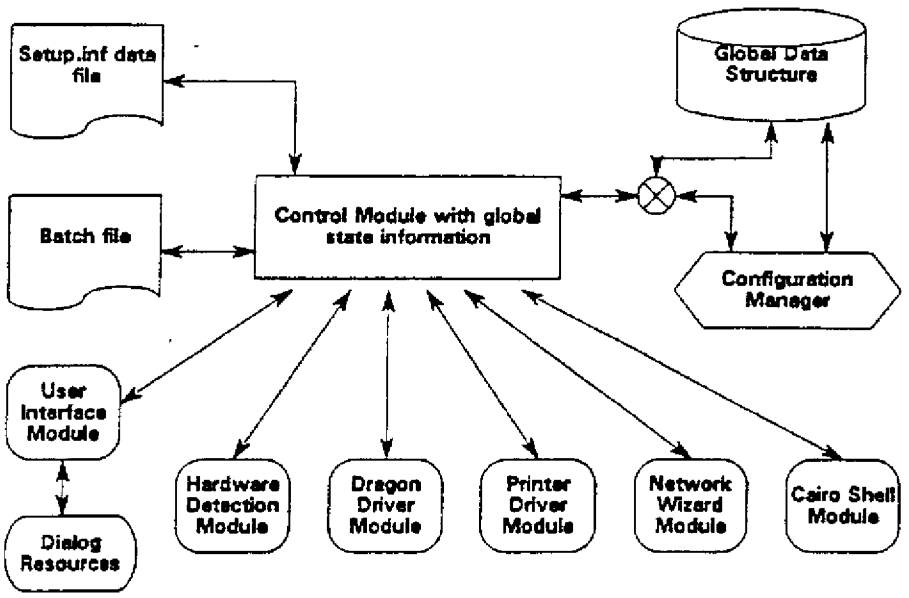

Setup.ini file
Setup.inf, setup.ssh, oemsetup.inf
Setup.ini file
DOS main Exe with integrated UI and many other routines
Win CUI main Exe with UI and many imbedded routines
Win GUI main Exe with UI and many imbedded routines
DOS GUI main Exe with integrated UI and many other routines
Static library with UI, hw detection, and many other routines
Second static library with UI, hw detection, and many other routines
Third library with UI, and other duplicated routines
Static library with UI, hw detection, and many other routines
Logging
Logging
Uninstall
Hard Disk

3.3.2. General Architecture for Chicago Setup

In order to meet the feature and project goals in the sections above, the Chicago setup programs will be built with modular components. Many of the routines in the current Janus setup programs will be extracted and reorganized as modular, dynamic link libraries. Library routines which are needed in both real and protect mode will be implemented using a new DLL architecture created for the Jaguar utilities so that the same code can be used in both modes. Standard interfaces will be defined so that other groups can supply modules to perform specific tasks, such as updating Win 3.X printer drivers. The user interface will be contained in two or more modules (depending on whether the Win GUI interface is up and running) so that any changes to the UI will be completely isolated to these modules.

This general architecture is illustrated in the diagram below[3]:

[3]Note the detailed API design has not been done, this figure is only for illustration of how the modules will work together in general.



3.3.2.1. Control Module

The heart of the setup program will be control modules working with various action modules. Control modules will most likely by specific to each phase[4] of setup and will provide for overall program control. As part of this, control modules will load action modules, as DLLs, and will provide certain common functions. Which subset of a predefined set of setup operations, the names of the DLLs themselves, all defaults for user input and all information about Chicago components installed will be kept in ASCII .INF files. The control module will call specific action modules to perform a unique operation, like setting up NLS support. Each of the action modules will utilize the data in the .INF files and results of user input (or batch file override) to determine the specific actions they will take.

When the user runs Setup, the control component will load, initialize, and then select and load a user interface component. The control initialization will include handling command line switches, environment variables and loading SETUP.INF. It will set up default global variables for things such as keyboard, display, network, etc. Then, it will load an initialization component do initial system checks. Note that the initialization and/or control component and/or the SETUP.INF file will be different for the upgrade, OEM packaged product, and OEM post install setup programs. The control component will then go through a subset of all possible operations as dictated by the data in SETUP.INF and possibly a batch file.

Besides loading and calling the various components, the control component will provide a number of services, including global variable handling, access to SETUP.INF and access to other components. For instance, it will provide a way for the file copy routine in the disk component to call the user interface component to update the gas gauge and to put up messages such as version conflict verification.

[4]The setup program will have up to five phases of execution, depending on whether it is running on an old operating system, on jaguar, on cougar, etc. These phases are covered in the Design section, below.

3.3.2.2. Global Data Structure

The global data structure will contain defaults for all user interface and /or detection results. These will be specified in SETUP.INF. It will also maintain an action list. The action list specifies the files to copy, move, delete, as well as other modifications to the system such as master boot record write, CONFIG.SYS changes, AUTOEXEC.BAT changes and *.INI changes. The global data structure will be written to the disk prior to any changes being made and then updated as the actions are carried out with the current status of the Setup. This will facilitate error recovery (setup continuation) and uninstall. There may also be some global Booleans used by multiple modules, including such things as Custom/Express install, and batch install. Existing code will be used as much as possible (including the file copy and action list maintenance code in Windows Setup).

3.3.2.3. User Interface Module

The user interface component controls all input and display. All displays will be specified by a standard Windows resource. The functions provided will be a subset of the Windows dialog box functionality with some additional user interface routines specific to Setup such as gas gauge control. When running under the Windows GUI, the user interface routines will utilize it. When running in "real mode", they will use routines written to modify the display directly. The user interface will detect if Setup is running in batch mode and return immediately with results of the specified dialog when specified by the batch file.

3.3.2.4. Other Modules

A description of each of the modules will be distributed for review. Examples of the other modules include the hardware detection module, the file copy module, the printer driver update module, the NLS module, the network wizard module, and the file system conversion module.

3.4. Setup Design

<u>3.4.1. General Code Path</u>

As stated earlier, a goal is to share as much code as possible between the upgrade, OEM packaged product, and OEM post install setup programs. This is possible and makes good design sense because these three programs share many of the same features and therefore can share many of the same components. Therefore, a design objective is to confine the unique characteristics of these three processes to the front-ends of these programs, so that the remaining majority of the program flow and code is identical to both. This approach allows us to place the proprietary Upgrade Setup technology at the front of the upgrade setup program so it can be readily excluded from the Setup source code licensed to OEMs.

The user interface part of the design is included in a later section. This section covers the basic code path and program flow of the control module and action modules.

The general Setup program is logically divided into five distinct phases, summarized below:

Phase 1: Collect data under old DOS/Win versions.

- Run under previous operating system
- Examine system hardware and software, see if can upgrade to Chicago
- Warn about bad TSR, drivers, apps, etc.
- Run network wizard to obtain information needed to reconnect net under cougar
- Determine best estimate of target system config and file copy requirements
- Load a "helper" program to manage interrupt vector addressing and provide other services which will help load and configure real mode device drivers after reboot in Phase 2
- Save uninstall and state information
- Copy minimum files to bootstrap system into cougar in Phase 3
- Minimal UI, move on to next phase as quickly as possible

Phase 2: Reboot to Jaguar and ensure no conflicts.

- Verify hardware access for critical components
- Read state information from hard disk
- Prepare hard disk for upgrade (partitioning, formatting, compression as needed)
- Use helper program to configure real mode device drivers, collect information on hardware resource usage, and to prepare for transition to protect mode
- Save uninstall and state information
- Minimal UI. Should be a very fast phase.

Phase 3: Transition to protected-mode and bootstrap core Chicago system.

- Read state information from hard disk
- Transition to real GUI supported by GDI and User, UI transitions as well
- Use special drivers to catch and handle error conditions
- Use a combination of real mode and p-mode drivers as necessary
- Use net wizard to reconnect to net (if applicable)
- Load and initialize the Configuration Manager, module which provides services to aid in installation and configuration of non core components (and apps tool).

Phase 4: Install all non-core Chicago components.

- Use Configuration Manager to install printer drivers, fonts, optional components, etc.
- Cairo operations such as building object Klasses, object properties, processing user data files to generate index files, etc.
- Install WOSA agents and the like

Phase 5: Give user chance to set user preferences, run optimization utilities.

- Control panel utilities: shell configuration and preferences, printer setup, network setup, memory optimization, disk defrag, etc.
- chain to other Setup programs.

Detailed design documents for these phases are available upon request.

3.4.2. Building the Three Setup Programs

Using these phases as building blocks, we can construct the three setup programs pretty easily. The Upgrade setup program includes all five phases. The OEM packaged product setup program includes phases 2-5, since we have to boot off the floppy on a naked system anyway. The OEM post install setup program includes phases 4-5, perhaps will an additional clean up module at the end.

3.4.3. User Interface Design

The design objectives for the user interface design are as follows:

- Make the user interface graphical
- Make the user interface entertaining and cool
- Do something so the user doesn't finish with setup and not know how to use the system or have some idea about the powerful features of the system (this is a big problem with Win 3.1, especially with OEM post install)

The detailed UI design for setup has not been fleshed out yet. After this design approach is reviewed and approved by upper management, detailed UI design work will start. However, at this point, these is one overall approach we would like to propose to address the third point, above:

- Scan the hard disk/ask some simple questions to determine user's knowledge level, how Chicago will be used. For example, novice, mostly dos apps or experienced Win user, mostly Win apps.
- Use this user profile information to select from a number of self-contained, autodemo type modules which cover just one basic concept such as how to start an application in Chicago. Factor in estimated time for setup to complete (net vs floppy install, options selected, etc.)
- Suggest these modules to the user in priority order, allow the user to deselect any or all modules
- Show modules running while setup is copying files and doing work in background. Only present setup UI such as questions, error conditions, change floppy requests between modules.
- Have some kind of default UI which shows exact setup status and progress for people who setup Chicago on a number of machines (option to show this in batch mode).

Could use this approach in all three setup programs.

3.4.4. Configuration Manager

As part of the Chicago Plug and Play design, the idea of a Configuration Manager has been proposed. The Configuration Manager provides a set of services which can be used to query and change system configuration and install and uninstall non-core system components and applications. The major benefits are providing a consistent programmatic way of performing these configuration operations and, for the first, time, a way to uninstall components and applications. Again, to leverage the use of the same code, we want to use the Configuration Manager to install the non-core components of Chicago and rely on the Config Mgr services to provide smooth and safe uninstall capabilities. A separate spec which goes into more detail is available upon request.

3.5. Requirements on Other Groups

- Conversion utility for partitions created by 3rd party utilities offering support for >1024 cylinders
- Backup/restore program which will run under DOS 3 2 or higher and Win 3.0 or higher
- Conversion utility which will convert from Stacker volume to Magic Drive volume and back again
- Conversion utility which will convert from Superstor volume to Magic Drive volume and back again
- Wizard for each supported network (LanMan, Winball, Novell, ??)
- We need to build a list of components we need at each point in setup and understand the implications. For example, just after we transition to protect mode and the system is only partly installed and in a transient state, some of the embedded drivers in the kernel components could cause problems.
- Need to have Dragon components follow the TBD requirements of the Configuration Manager when installing. They should be using the Config Manager API to access the registry to store and retrieve data.
- Setup program needs a piece of code/mechanism to detect/indicate if it is appropriate to load a given VxD when bootstapping Chicago for the first time.
- Need to have each kernel component, ie., DOS386.EXE, log its loading and initialization via a TBD API so setup can recover from errors, provide diagnostics, and perform an uninstall
- Need to have Jaguar kernel support sticky drive letters so we can preserve drive letter mapping of original system
- Setup needs to be able to trap and handle GP Faults using a high priority VxD
- Modules which other groups supply need to be designed so that minimal UI is done before Windows (User, GDI, Kernel) is up and running
- Everything needs to be version stamped. All Chicago components, all third party Chicago enabled files, etc.
- A list of the modules we need other development groups to write will be forthcoming. Examples include a module to update/install printer drivers, a module to update/install the Cairo Shell, etc.
- Chicago needs to work if installed in multiple directories (assuming we support this feature). Need to understand requirements/limitations here.
- Better A20 delay detection (HIMEM.SYS)
- NLS utility (keyboard must support password protection)

3.6. List of Open Issues

- In the case of ungrading a Chicago system to a new version, not sure how to replace files which are in use, need to work out an approach.
- What hardware do we need to detect? We need to generate a definitive target list and have people review this, reach agreement.
- There are a number of open issues related to the case when Setup replaces a real mode device driver with a p-mode driver. How to handle in Super Exclusive mode, ensure we don't install files to any storage devices controlled by this driver, etc.
- We need an approach to handle upgrading and the first time install for diskless workstations
- What help engine and model will we use for setup? May differ, depending on which section we are in.

3.7. Next Steps

- Need to have general design and architecture as well as the related outside dependancies and open issues reviewed by the Chicago development team and upper management. Mail on this will be forthcoming.
- Need to reach closure on the list of features and their relative priority for this first pass. Features will be added over time, but we need to take a best guess at what will be in the product for the first beta
- Work with the others on the Chicago team to produce and reach agreement on the related design specs (listed below)
- Produce a schedule which takes us to the first beta milestone

3.8. List of Related Documents

Many of these documents can be found in the \\johnpa\setup (readonly) sharepoint.

3.8.0.0.1. **Chicago Setup Project Overview,** \project.doc

Covers main goals of the setup project, basic requirements, general approach we will take in design and development

3.8.0.0.2. **Plug and Play in the Chicago Product,** \pplay\pplay1.doc

An overview document on what features and components of chicago are part of the plug and play group.

3.8.0.0.3. **Requirements on Config Manager,** \confman\confman1.doc

Covers proposed design requirements on the config manager and steps through an example of how a system component should be installed and uninstalled using the config manager

3.8.0.0.4. **Detailed design document(s) for Phases 1-5,** *work in progress*

Early drafts of phases 1-3 are available on request. Phases 4-5 are yet to be really thought through in detail and have strong dependancies on the shell model.

3.8.0.0.5. **Detailed design document for the file copy/version checking engine,** *work in progress*

Early drafts are available on request.

3.8.0.0.6. **Design document for processing configuration files,** *work in process*

Spec for the engine which will process autoexec.bat, config.sys, *.ini files, modify them, extract information, etc.

3.8.0.0.7. **Hardware detection library API spec,** *work in process*

We are building a hardware detection library from various sets of code. The control panel and some of Tomle's hardware plug and play modules will need access to this library. This spec will propose a general interface which other Chicago components can use.

3.8.0.0.8. **Interface to User Interface Module,** *to be written*

Will cover proposed interface between setup core engine components and a single user interface module.

3.8.0.0.9. **Interface to Network Wizards,** *to be written*

Will cover the proposed requirements for a Network Wizard (as far as the setup program is concerned) and also a proposed interface.

4. Plug and Play in the Chicago Product

4.1. Introduction

Plug and Play is a term which is used by a lot of people at Microsoft to mean a number of different things. The purpose of this discussion paper is to nail down what Plug and Play means for the Chicago product. This topic has been addressed in the Chicago Strategy Document written by David Cole and more recently by Ralph Lipe in a memo (included in the appendix), and much will be drawn from that work.

We want to use this paper as a discussion guide for meetings with Bradsi, Davidcol, Dennisad, and others on the Chicago team.

A section on specific scenarios which we will handle needs to be added. Work in progress, look for this in future drafts.

4.2. What is Plug and Play in the Chicago Product?

Plug and Play for Chicago is as much a design methodology as it is specific features. (Ralph's memo touches on the methodology) This section lists the key high level features and goals of plug and play and then describes the common design points.

- Easy and safe to setup and configure Chicago on a running PC, with DOS or DOS and Windows
- Chicago easy and safe to configure/customize on an OEM pre-installed machine and on a system which is up and running.
- Easy and safe to set up and configure hardware on existing PC's.
- Transparent installation of hardware on future PC's designed to run Chicago
- Applications and optional system components software easy and safe to install and uninstall
- User preferences and system configuration data used to help user change their system. The system remembers and makes use of available data to configure itself so the user doesn't have to. If the user installs a DOS app, DOS VM configuration information is automatically associated. On a networked system, option to have user preferences follow users from machine to machine.
- It is safe to plug laptops into docking stations and PCMCIA cards into sockets without cycling the power or rebooting the Chicago system. Easy and safe to configure the new devices which are available.

The repeated themes are easy and safe, did you notice? Easy to do any kind of setup and configuration; easy in the sense that a novice can do it and *enjoy it* and an experienced user can do it and enjoy it. Easy in the sense that the system is far more usable. Changing screen colors or video mode is a snap. Plugging in a new board is a breeze. Usability goes far beyond plug and play, of course, but making setup and configuration operations more usable is a large part of plug and play. Easy also in the sense that the system does as much of the work for the user, especially for the novice who prefers to work in this way.

Of course, since easy means different things to different kinds of users, we will have to cover all bases in Chicago. For novices who don't want to have to learn about technical details, our plug and play features will need to hide this and present choices in very simple terms. For experts who want ultimate control, we will have to offer this option. For those who perform setup and/or configuration operations on a number of machines, we will have to find ways to make these operations easy, fast, and with the amount of control required and desired.

Another large part of plug and play is safe. Safe in the sense that a user doesn't have to every worry about rendering their system unbootable or unusable. Safe in the sense that the user always understands what is happening and is not scared of operations which seem mysterious. Safe in the sense that attempting a setup and/or configuration operation no longer means there is a pretty good chance they will be forced to go get a friend to help or make a call to PSS, even *after* reading the documentation.

4.3. Elements of Plug and Play in the Chicago Product

In previous discussions, David Cole has grouped the plug and play features into the following categories:

1. System setup
2. Hardware setup and configuration
3. Configuration and maintenance operations on a running system
4. Configuration operations based on dynamic events

These four areas share some common features. First, the user ALWAYS will be able to reboot the system back to a known good state, should they ever hang their machine or incorrectly configure it (for example, installing a bad device driver). This means that whatever the setup/configuration operation the user was preforming, if they want or need to restart their machine and have it come up in a known good configuration, ie with a known good video driver or a known good net redirector, they will be able to do this. Second, the UI for setup and configuration operations will be consistent along key dimensions which are critical in usability tests for easing the fear factor/making things seem more familiar and for making complex operations more usable in general. Third, as stated previously, as much as possible will be done automatically for the user. Wizards will be available to novice users who opt for this approach. Finally, an effort will be made to make configuration and setup operations fun, entertaining, and "cool."
Each of the four general areas is covered in more detail the sections below. The key benefits of plug and play and the implementation approach for each specific area are discussed.

4.3.1. Setup

The key benefits of the setup[5] which relate to the Chicago Plug and Play goals are:

- complete uninstall of Chicago system
  - base system
  - optional components
- express vs custom modes of operation
- easy to use, batch file operation
- seamless operation over a network
- seamless operation with Hermes
- full support of important licensing schemes[6]
- previous user preferences retained as appropriate, with option to change any of these during setup
  - On same machine
  - Transferred from old machine to new machine
- option to view entertaining segments of video, sound, autodemo stuff during disk copying which highlight important messages. This is important for getting new or infrequent Windows users to stick with the Chicago product during the first 30-45 minutes of their experience, especially in the OEM post install case. Jeffl, Timbre, Marionj, and Chriswo are already thinking about content and message issues for these pieces.

4.3.2. Hardware Configuration

- Software to help configure existing and/or new hardware
  - Configuration logic to track and allocate hardware resources, handle conflicts and error conditions
  - Configuration logic to install and uninstall device drivers
  - Wizards: To help users over the rough spots of configuration, since it is unlikely we will cover 100% of the old architectures, we will need to create wizards targeted at building configuration information files.
  - Configuration check of hardware at every system boot.
- Software Standards
  - Make it easy and safe to install or remove existing devices from a single configuration point
  - Use of EISA and MCA .cfg files for configuration information
  - Design IHV provided load instructions for device drivers. Tells configuration code how to build a config.sys or system.ini line.
  - Include ability to run IHV provided installer (as option or at request of Wizard)
  - Make hardware detection code extensible so code can be added for new devices

[5] I am not going to make the distinction between upgrade, oem packaged product, and oem post install setup right now, you can make correct associations yourselves. Will clean this up as needed later.
[6] At this point, we are not sure which licensing schemes will be important in the Chicago timeframe. The LS API developed by Cairo is great, but apps have to be LS API enabled before this will work. We expect schemes which do not require the customer to upgrade their apps to also be important and popular in the Chicago timeframe. People in the apps interoperability and SWAT groups share this view. If people have more information on this, please forward it on.

- Standard way for a device driver to be installed and uninstalled
- Hardware Standards
  - Make it transparent to install devices in future machines. This means new designs for future PC's as well as new designs for add-in boards, and peripherals. Example, what is the standard way we will detect a video card and monitor in a new machine?
  - Work with Compaq and other OEMs and IHV's on Plug and Play capable machines
  - Address the problem of new board or peripheral configuration in old machines
  - Address the problem of old board or peripheral configuration in old machines. For example, develop a software method for slot addressability on an ISA bus.
  - Design a standard reporting mechanism for hardware configuration information
- Evangelize hardware and software standards to OEM's and IHV's

4.3.3. Software Configuration and Maintenance Operations

The important goals for this section are:

- simple UI and programmatic paradigm for installing and uninstalling any software except for base OS files. Any software includes optional system components such as printer drivers, fonts, applets, etc. and also any applications software. Way to handle old apps and system components.
- Look through the existing configuration processes in maintenance mode setup and the control panel and remove hard to answer questions, improve overall flow and UI design. This is the low hanging fruit which will reap a large return.

  Video - User must today indicate the chip set manufacturer and the amount of installed memory on their video boards. Even then, once the user defines the board and picks a video resolution, the Monitor may not be capable of displaying at that resolution.

  Rather than presenting a big list of combinations, as we do today, we could show the options graphically with a diagram of a small screen, the font sizes, and a rainbow or the like.

  Keyboard - Today the user is asked to identify their keyboard by the number of keys. Can you get the image of a user counting 1,2,3... Again , we can do a lot to enhance the UI here by showing show some thumbnail sketches of keyboard layouts and have the user select one. Only if further clarification is needed by the user pressing keys or the like, we can then ask for this.
- express vs custom or simple vs detailed modes for many operations
- replacement of a real mode device driver with a VxD
- full support of popular licensing schemes
- Memory optimization, including UMBs if DOS apps are used
- Hard disk defrag, backup, virus scans
- All operations currently in Win 3.1 control panel and in maintenance mode setup
- As described earlier, a complete review of the UI for all Win 3.1 configuration operations so that the minimum is asked of the user

4.3.4. Dynamic and Event Driven Configuration

What can best be said about dynamic configuration is that it has not been thought about much. The concept is generally described as a system configuration action taken in response to a system event.

- When a PCMCIA device is plugged into a socket, easy or seamless way to make whatever resources are on this card available to the system. This includes configuring the card, given the current configuration of the machine and the available resources. Easy to recover or block use of resource(s) if conflicts arise. Similar set of issues if device is removed from the socket.
- Same for a docking station, making a new set of cards, hard disk, floppy drive, external peripherals, and video available to the machine in an easy or seamless way. Easy to recover and/or reconfigure if conflicts arise.
- Coming into and going out of range for a wireless network

4.4. Pieces of Plug and Play

The major pieces of plug and play are listed in this section, as envisioned today. These pieces are listed and briefly described here so everyone can think about the goals and features listed above in terms of deliverables.

1. **The upgrade/OEM packaged product setup program.** A modular program with 4 or 5 distinct phases of execution. Engine components such as file coping, file system conversion, etc. implemented as dll's which can be run in real mode and in protect mode, so many can be used as engine components for control panel functions as well. In the upgrade case, complete uninstall capability is provided.
   Phases 1-5 of setup and basic support modules. There is a lot of work here: control modules for all five phases, basic support modules such as file copy, file system conversion, uninstall of core components, etc See the setup spec for more detail.
   Network upgrade wizard. Will aid in the upgrade of a network by obtaining network login scripts, identifying the network installed, identifying the current network connections, etc. Hopefully, this will come from the network group.
   Printer driver upgrade module and other modules. As covered in the Setup spec, we want the groups responsible for writing printer drivers and other distinct components to also supply the install/uninstall modules for these components. This way, the group which knows the most about a given area builds in easy install and uninstall from the beginning so the setup team doesn't have to be an expert in all areas.
2. **The OEM post install program.** Very similar to the last 2 phases of the upgrade setup program which install optional components, set user preferences, and give the user an option to run a tutorial or autodemo. Many of the same modules will be used.
3. **The configuration manager.** The piece of code which gives access to system state and configuration information and which provides services to install and uninstall programs. Using these services, all non-core system components and applications software can be installed and uninstalled using the same UI and programmatic paradigm. The configuration manager is used in the last two phases of the upgrade setup program. The configuration manager is also for providing the last known good boot state, should the user need to revert back to this state.
   Hardware configuration helpers. This is a set of routines accessible through the configuration manager which provide configuration services. Included would be; Return an available configuration given a set of configuration metrics[7], enumerate all free H/W resources, etc.
   Dynamic configuration logic for config manager. Code to control actions taken by the system in response to some dynamic configuration event. This assumes the existence of an event manager which can notify the configuration manager when an event occurs.
4. **Configuration wizard.** The configuration wizard will help a user pull configuration information from their H/W manual. The wizard will build this information into a partially complete .cfg file for use by the remainder of the Plug and Play modules.
5. **Hardware detection.** An extensible library of hardware and device driver detection routines which will be used by the setup program, the hardware configuration pieces, and possibly some of the control panel pieces.
6. **Hardware configuration logic.** Set of one or more routines which handle hardware configuration tasks.
   Install new hardware button logic. Control procedure which walks the user through the installation and configuration process. May invoke the configuration editor or one or more wizards.
   Boot time hardware configuration logic. Code which scans the hardware and configuration files for changes since the last boot. If things have changed, the hardware configuration logic kicks in.
   New methods for configuring old add in and peripheral hardware in old machines.
7. **Configuration tools for control panel.** The engine and UI code for the control panel functionality.
   Basic UI work. Work to simplify and enhance the UI so the user is only asked to select from very simple and consistent choices.
   Configuration information entry utility(s). This is one or more utilities the user can invoke to force manual overrides on system configuration or installation settings. Functions to be provided would include:
   - Manual edit of software configuration settings (ie, config.sys entries and the like)
   - Manual configuration entry which allows a user to build or edit a .cfg file from scratch.
   - H/W Configuration editor which allows the user to set or change a devices configuration. These will be error checked to ensure the user does not create conflicts.
8. **New Hardware design and evangalization**

[7]A configuration metric is defined as the range of settings a configurable card can be set to. A card which can be set to IRQ 5,7 or 9 has the set {5,7,9} as its IRQ metric.

CMS 00013739

RBC 04140

Plug and Play PC design.
New Plug and Play add-in board and peripheral designs.

9. **Fast Transition Code.** Code which works with the configuration manager and other pieces to handle these two special cases:
   Docking station transition code. Handles the transition in configuration which takes place when a PC is plugged into or removed from a docking station.
   PCMCIA transition code. Handles the transition in configuration when a PCMCIA card is inserted or removed from a socket.

4.5. Open Issues

1. What can we decide we definitely will not do in the Chicago product?
2. What help will we get from the Rover group on the Dynamic Config stuff?
3. How important is consistency between Chicago and NT? The concern here is that we don't want to make NT look bad.

4.6. Next Steps

1. Collect everyone's comments and incorporate into next draft
2. Review Apple system 7 plug and play functionality on the 24th
3. Once agreement on major points is reached, ensure these points are reflected with the correct priority in the plug and play feature specifications (a first draft of the setup and hardware configuration specs are done. drafts for the software configuration and dynamic configuration pieces are yet to be written).
4. Iterate, do more investigation as necessary to generate a list of work items and a schedule with key milestones.
5. Prepare for the Bill Gates review meeting on November 4th.

4.7. Appendix A--PC Mag's Quick Look Review of Chicago Plug and Play

In the spirit of thinking about what the review should say about what we are designing here, our idea of the ideal quick look review is outlined below:

System setup is fantastic

- Easy to setup, both for a novice and a computer nerd like me
- Setup handled difficult hardware and software configurations
- Network upgrading is handled very well, even from non-MS networks
- The uninstall process is smooth and makes the setup process very safe
- Sucking configuration information from my old PC saved me a lot of time!
- Option to view short autodemos about Chicago is valuable for everyone
- Setup itself is entertaining and "cool"

Plugging in new hardware is 10 times simpler with Chicago

- Old boards are easy with the hardware configuration wizards
- New boards configure themselves!
- Users never have to fend for themselves during hardware installation
- System is checked for changes at each boot, which makes it safe.
- Can always backup to a previous state if you get in trouble

Installing and UNINSTALLING software is really easy

- Installing an app much simpler. Dragging an icon to the "control panel" starts the process.
- Yes, you read right, if you are a helpdesk person, consider looking for a new job *now*. With Chicago, you can selectively uninstall applications or even pieces of applications, Chicago automatically keeps track of the configuration changes so you don't have to.

Changing my system configuration: just pick the picture you want

- Selecting a new video driver involves picking the miniature screen I want, not choosing from a long list of arcane driver descriptions
- Changing colors is a drag and drop procedure

Wow--Chicago understands docking stations and wireless networks!

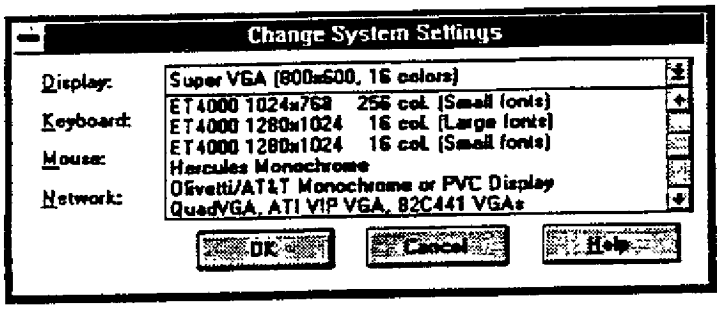
4.8. Appendix B--Ralph's Memo on Plug and Play

4.9. What the Hell Is Plug-N-Play (In the Chicago Product)?

"Plug-N-Play" seems to have a lot of people confused about how to approach the problem. It's actually very simple. It means we must do a few simple things to make sure that most things that require user intervention are configured automatically, plan for future hardware capabilities, and do a great job of setup and configuration. Here are some specific examples:

4.10. Display Drivers

Configuring video drivers is too complicated. We recently created a generic 256 color super VGA driver that supports lots of cards. This is a great step in the right direction. We need to the same thing with our VDDs and Grabbers. Ideally the drivers or a hardware detection library will detect the exact vendor/chip set for the card in the machine, and then only present the user with simple choices (such as number of colors and resolution) for configuring the display driver. Instead of being presented with a huge list of things like this:



Super VGA (800x600, 16 colors)

ET4000 1024x768 256 col. (Small fonts)
ET4000 1280x1024 16 col. (Large fonts)
ET4000 1280x1024 16 col. (Small fonts)
Hercules Monochrome
Olivetti/AT&T Monochrome or PVC Display
QuadVGA, ATI VIP VGA, 82C441 VGAs

The list would contain simply:

640x480 32767 col. (Large Fonts)
800x600 16 col. (Small Fonts)
800x600 16 col. (Large Fonts)
etc.
etc.
Other Display (Requires disk from OEM)...

Note that this new list presents only the things that users want to be able to change, and does not require and knowledge of the chip set, OEM, memory size, etc. It just shows all the valid choices you can make for this machine. *That's* Plug-N-Play.

We should create a VDD that supports every single VGA adapter out there. Our grabber should work for every single VGA adapter out there. If you can accomplish that then you've got a great story. We can ship all the drivers in our box since there's just one!

4.11. Mouse Drivers

Our base mouse driver will support Microsoft, IBM, Logitec, Mouse Systems, and other vendors mice straight out of the box. We will *not* ship separate drivers for each mouse type, our mouse driver will handle all of them. Our cursor drawing code will handle all displays supported by our generic VDD. No TSR will be required to use the mouse in a DOS box.

We will allow IHVs to write their own cursor drawing and/or mouse hardware code to replace ours. This will make it easier for 3rd-party hardware to be supported.

Even for machines that don't have a mouse when Windows is set up will get the mouse driver. The driver will always perform any detection tests that are non-intrusive and quick every time Windows boots. If someone plugs in a mouse, poof! The mouse works. *That's* Plug-N-Play.

4.12. Setting Up New Cards

By simply maintaining a database of which IRQs, UMBs, I/O ports, and DMA channels are in use, the system can make setting up an existing card much, much simpler. A wizard can help walk the user through the steps of selecting jumper settings while avoiding conflicts with other devices in the system. It's not perfect, but it's one hell of a lot better than the situation you have today where your machine gives you no help in attempting to determine how the card should be configured. All of the informatio about IRQs, DMA channels, etc. is in there, you just have no way to get to the data. Simply giving the user access to this information is 1/2 the battle. Vendors could ship new setup programs with existing hardware that could go as far as drawing a picture of the card on the screen and showing the proper jumper settings.

For new hardware, we need to work closely with IHVs to define a new configurable card standard. This is a long-term bet and it won't yield fruit immediately. However, if we do it right, 3 years from now all new cards will be auto-configurable so you'll just stick it in the machine, the setup program automatically pops-up when the system boots and copies the correct driver and configures it properly all without user intervention. ***That's*** Plug-N-Play.

4.13. Configuring Modems and Printers

Today, users are asked to select which port various devices live on. Forget about it! They don't have any idea. We need to write detection code that checks for a Hayes compatible modem on all COM ports. If we detect one modem, then that's how we automatically configure Terminal. For printers, we can simplify the choices by first eliminating all non-present ports from the list of selections (for example, my machine has one LPT port, but I am allowed to choose from LPT1, LPT2, or LPT3). We could do things like print "THIS PRINTER IS CONNECTED TO LPTx <page eject>" to all LPT ports. The user could then easily find out what printer is on what port. In this case we can't automatically find out all of the information ourselves, but with some help by a novice user, we can make it work without a call to PSS. ***That's*** Plug-N-Play.

4.14. Other Things We Can Do

There's lots of low-hanging fruit out there. Sometimes it's just so damn obvious that we overlook it. Why are things hard? Why can't people figure out how to run DOS apps in the background? How can you fix it?

Everyone should carefully think about what type of user configuration is required for their piece of the product. If a dialog box is more about asking users for information than presenting valid choices to them then that's not Plug-N-Play -- That's Plug-N-Pray.

5. Draft Shell Items

5.1. Introduction

This is a list of things that need to be done and areas that need investigation (from Chrisg). It is still under review by the team. For a shell spec. please refer to the Cairo documentation (available from Stevem in Cairo group). Chicago plan is to take Milestone 4 from the Cairo team in late October '92 and use that as the basis for our shell.

5.2. Common UI Code:

5.2.1. Percent done dialog.

5.2.2. Implement file.run common dialog.

drop down combo MRU
support all show parameters (store with MRU entries)
allow setting working diretory, browsing for working dir.
other actions besides run on associated files (print, view).

5.2.3. Insert Disk with browse.

save previous locations where file were found (network share, locations on hard drive).

5.2.4. Setup copying files UI with graphical progress (disks stack up, etc).

5.2.5. Exit Windows common code (includes restart/reboot/shutdown).

5.3. More UI Related code:

Cairo style drag/drop.
Magic window tile code (position based).
Sensible auto-arrange.
Window management app that auto-tiles, auto-minimizes, etc.

5.4. Setup/Config common code:

5.4.1. System schemes.

start with our color schemes
get SoundBits sound schemes

5.4.2. INF file parse.

5.4.3. Copy with version checks, deals with files in use (does ExitWindowExec()).

5.4.4. Ini file changes with commit/backout.

5.4.5. Data driven configuration code.

5.4.6. Printer install/plug & play.

Postscript: auto-detect PS pritners on COM ports
Postscript: auto baud rate detection
Posotscript: model/VM detection
test ports, LPT/COM (test page print, walk user through problems)

5.5. Random thoughts

Drag/Drop CPL files into control panel (copy to sys dir, install)
Icon selection/hilight, not just title.
Taskman, hide(minimize) everyone.
Run install programs automatically when floppy inserted.
COMMDLG printer setup, Unlisted Printer... option that goes to printer install code.
GDI support for loading drivers over the net (full path name to drivers stored in win.ini).
Searching for EXEs that move on the disk, that aren't on the path when doing file.run...
Printing monitor app. Small topmost window that shows printer status, allows queue manipulation, deleting/pausing jobs.
More robust error reporting for WinExec()/LoadModule() that will let us get a list of DLLs that are missing.

5.6. Task Manager

5.7. Control Panel
cache CPL icon bitmaps for fast load-
convert to use view control

5.8. Control Panel.Color
share sample drawing with Window Metrics
share schemes management with Window Metrics
support drag/drop color settings
256 color display solids

5.9. Control Panel.Fonts
spec \windows\fonts directory (GDI changes, setup requirements, control panel)
About... box for TT fonts
view by family/type
user defined sample/print a sample
support uninstalling fonts temporarily, preview before install
font "schemes/sets" that can be installed/uninstalled
talk to spag guys for more ideas here.

5.10. Control Panel.Mouse
make this more generic "pointing device".
status info: mouse settings, relative/absolute, com port used
change/install mouse from maintence mode setup
Setup... button to configure
mouse "schemes", named settings (new user, power user).
improve the visuals for the mouse diagram
new dialog with all mouse controls from sandeeps

5.11. Control Panel.Ports
use listbox for picking ports
diagnostics, status on ports (like printers, connect...)
support for more than 4 com ports
make settings clearly marked as for printing

5.12. Control Panel.Keyboard
add "cursor blink rate" from desktop
change keyboard type from maintence mode setup

5.13. Control Panel.Wallpaper
add "pattern" support, from desktop
add other allignment options besides center (top, bottom, left, right).

5.14. Control Panel.Printers
evaluate simplifying this dialog. mergeing with print manager.

printer connections show net names
point and shoot printer install/configure with winball servers

5.15. Control Panel.International

GerardZ's new NLS and configuration stuff.
Multiple Keyboar layout

5.16. Control Panel.Date/Time

add date/time formats from internatioal to this dialog

5.17. Control Panel.Network

must be able to change installed network if the net driver does not support this

5.18. Control Panel.for Multimedia

5.18.1. Midi mapper

have MM people completely redesign this

5.18.2. drivers

make this a MM specific "Multimedia Drivers"
make this easier to install all drivers for specific hardware (install "Sound Blaster Pro" installs all 3 drivers)
re-code this using our setup/configuration common code.

5.18.3. sound

make this use the common file open dialog for browsing.
support "set defaults" button.
sound schemes.
heirerarcical sound definitions.

5.19. New CPLs:

5.19.1. MS-DOS Application settings.

parameters from current win386.
global PIF type settings.
status on currently running DOS apps (memory usage, etc).

5.19.2. System Hardware

list CPU and speed, physical ram, # hard drives/type (scsi, esdi, ide),
net card, com and lpt ports, cd-rom drive

5.19.3. Display Setup/Configuration

replaces maintence mode setup
measure the display
set device dimensions (log pixels, physical pixels)

5.19.4. Setup add/remove windows components

5.19.5. Setup applications

5.19.6. App install/deinstall control.

5.19.7. Docking station applet for chaning displays, printers, network shares.

5.19.8. CD-ROM cpl. Enables/disables, installs MSCDEX and CD-ROM device drivers.

5.19.9. Handicapped support/configuration for access drivers.

5.19.10. Power managment.

5.19.11. Pen CPLs, merge functionallity

make all CPLs deal with multiple instances.

5.20. Commctrl

5.20.1. toolbar

large button support with text
indeterminate state support
fast drawing/bitmap for up/down state.
use DrawFrameControl() to get new 3d.
review custiomization implementation/messages.
use BTTNCUR.DLL

5.20.2. trackbar

TBS_AUTOTICK support for big ranges.

5.20.3. spinner

subclass buddy for keyboard interface.

5.21. File Manager

background search
use dos file find engine for search
clean up confirmation, allow "yet to all" confirm for read only files.
general purpouse graphical status display for us in progress dialog.
diskcopy UI needs rework, multiple copies, write image to disk, etc.
format dialog UI, status bar on dialog, cancel button, chose drive, multiple formats, auto disk insert detect, etc.
file copy improvements, better status display, detect file won't fit on dest.
smaller reads for smother multi-tasking.
use reg db for per file type actions "Play", "Edit", "Print", etc.
change "Associations..." to file type/action editor.
support actions on programs, directories, NULL extension files.
manual file type override support.
new drive icon for 3.5 vs 5.25 drives.
Status gauges for all status windows.

5.22. Print Manager

do auto install when connecting to a net printer. work with sparta guys on this.
defered/metafile printing support (viroont).
save & restore state on shutdown.
large button toolbar with text
clean up server quereies, add "breathing" between net calls.
review banner page generation, merge with metafile printing.
context popup support.
printer on desktop control.

5.23. Item Manager

implement filetype/action in item manager.
decide on grp file format.
> 64k bitmaps
integrate PIF editor for dos apps.
edit file type actions as in file manager.
context popup support.
drag/drop server for files, to desktop, etc.

5.24. Taskman

Arrange windows (in addition to cascade and tile)
Use group control
File.Run
Exit Windows

5.25. Dos App Configuration

5.25.1. Buttons for PIF file settings.

Progman (shell) calls DLL to config dos apps.
Winoldapp uses DLL for config of running apps.
PifEdit uses DLL for editing/creating PIFs.
Auto pif file creation for setup.
Default DOS app config control panel applet.
Sys VM multitasking settings.

5.26. Multimedia enhancements

5.26.1. New Media Player with OLE support.

5.26.2. Sound Recorder enhancements.

Cut/Paste sound (partialy supported now).
Cmd Line parameter to play sound (don't display).
Use the trackbar instead of scroll bar (UI tweaks).

5.26.3. CD Player app (musicbox).

5.26.4. New MCI drivers.

AVI (Audio Video)
Foghorn sound driver.
Other Popular sound card support.

5.26.5. AVI windows tutorial (for CD rom release).

5.27. Windows Tutorial/Learning Windows

CMS 00013748

HIGHLY CONFIDENTIAL

MX 2120454
CONFIDENTIAL

RBC 04149

6. GDI

6.1. Goals

There are just a few goals of any improvements to the graphics layer of Windows 3.1: Plug 'n Play, reducing system limitations, performance, size, 16-bit display drivers, 24-bit display drivers, 32-bit display drivers, making device drivers easier to write, device independent color, and making GDI more consistent so ISV's and end users won't be surprised by random behavior. Compatibility is a strong constraint, nothing is added that leads to backwards compatibility problems; nothing gets fixed for fixing's sake. 32 bit code is not a goal, it is a tool, and is added only where needed.

We will leverage existing code and expertise in other groups as much as possible: NT, Jaws, SPAG, MultiMedia, applications, third parties, etc.

6.2. Schedule

This schedule assumes a Sept 1993 shipdate (I don't care what the official ship date is), which means code complete Feb 1992, which means that there are 8 calendar months for development: July to Feb inclusive. By 'code complete' is meant that the code completely meets spec, that it has been completely and accurately code reviewed, that the reviewers and management want no more changes, the code has been well tested by the developer, and there are no known bugs in the code. No group at Microsoft has been able to accurately predict ship dates based on estimated ccompletion dates, so I will simply use the scheduling technique of putting a developer on a component with lots of time. If the component doesn't take as long as expected then the "nice to do" category will start to get implemented. It makes much more sense to add stuff to a schedule as stuff gets completely done, rather than trying to shove as much as possible into a schedule that is too tight, or spend time arguing what schedule features should be dropped. **Implementing the few key features and meeting the ship date is more important than a thousand features.** Also this is a very performance oriented release, I want enough time to tune, tune, tune! I want enough time to port to 32 bits those pieces that make sense as 32-bit code.

We also know that we are very much a support group as well as a development group. As new features get added, and these affect device driver writers, those writers can only properly be supported by developers. In my view we also need two people dedicated to display drivers because in the end we end up having to find/fix bugs in too many third party drivers.

Have to do		
Bezier curves	1/3 person	rayen
Paths	2/3 person	rayen
DIB driver	2 persons	amitc raypat
General performance	2/3 person	kensy
GDI local heap	1/3 person	kensy
DIC	1/4 person	davidw
Re-entrant GDI	1/4 person	davidw
Metafiles	1/3 person	chah-chi
Masked bitblt	1/3 person	?
3.1 postponed bugs	~100	everybody

Nice to do	
Cleanup	1 person
line layout	
multiplane	
transforms	

6.3. Have to do

6.3.1. Bezier Curves

This is added for performance reasons. With the change to do metafile spooling 'return to app' time will be faster if applications make use of beziers instead of passing down polylines. In addition printing to devices that support beziers will be faster (postscript, future revs of jumbo/dumbo). Beziers have been requested by many ISV's for a long time. Windows 16 presently has no real curve capabilities. This is a port of the NT code and APIs.

The two functions added are

PolyBezier	Draws Bezier curves
PolyBezierTo	Draws Bezier curves

6.3.1.1. PolyBezier

BOOL PolyBezier(*hdc, lpPoints, nCount*)

```
HDC hdc;               /* handle of device context                  */
LPPOINT lpPoints;      /* address of endpoints and control points   */
DWORD nCount;          /* count of endpoints and control points     */
```

The **PolyBezier** function draws one or more Bezier curves.

Parameter	**Description**
hdc	Identifies a device context.
lpPoints	Points to an array of POINT structures which contain the endpoints and the control points.
nCount	Specifies the number of points in the *lpPoints* array. This value must be one more than three times the number of curves to be drawn, since each Bezier curve requires two control points and an end point, and the initial curve requires an additional start point.

Returns

The return value is TRUE if the curves were drawn. Otherwise, it is FALSE.

Comments

This function draws cubic Bezier splines using the endpoints and the control points specified by the *lpPoints* parameter. The first curve is drawn from the first point to the fourth point, using the second and third points as control points. Each subsequent curve in the sequence needs exactly three more points: the end point of the previous curve is used as the start point, the next two points in th sequence are control points, and the third is the end point.

The current position is neither used nor updated by the **PolyBezier** function. The figure is not filled. This function draws lines with the current pen.

6.3.1.2. PolyBezierTo

BOOL PolyBezierTo(*hdc, lpPoints, nCount*)

```
HDC hdc;              /* handle of device context                  */
LPPOINT lpPoints;     /* address of endpoints and control points   */
DWORD nCount;         /* count of endpoints and control points     */
```

The **PolyBezierTo** function draws one or more Bezier curves.

Parameter	**Description**
hdc	Identifies a device context.
lpPoints	Points to an array of POINT structures which contain the endpoints and the control points.
nCount	Specifies the number of points in the *lpPoints* array. This value must be three times the number of curves to be drawn, since each Bezier curve requires two control points and an end point.

Returns

The return value is TRUE if the curves were drawn. Otherwise, it is FALSE.

Comments

This function draws cubic Bezier splines using the endpoints and the control points specified by the *lpPoints* parameter. The first curve is drawn from the current position to the third point, using the first two points as control points. For each subsequent curve, the function needs exactly three more points, and uses the end point of the previous curve is used as the start point for the next.

PolyBezierTo moves the current position to the end point of the last Bezier curve. The figure is not filled. This function draws lines with the current pen.

6.3.2. Paths

This is added for performance and sanity reasons. With the change to do metafile spooling 'return to app' time will be faster if applications make use of paths instead of passing down polygons. In fact there is no way to construct polygons that scale well unles. gobs of lines are used. In addition printing to devices that support paths will be faster (postscript, future revs of jumbo/dumbo). Paths have been requested by many ISV's for a long time. Windows 16 presently has no real path capabilities. This is a port of the NT code and APIs.

The functions added are:

AbortPath	Aborts and discards any paths in a DC
BeginPath	Starts a path bracket
CloseFigure	Closes a figure in a path
EndPath	Ends a path bracket
FillPath	Fills the current path
FlattenPath	Transforms curves to lines
GetPath	Returns all lines and curves in path
PathToRegion	Creates a region from a path
SelectClipPath	Selects current path as clip region

The following functions may get added, depending on time, they fall in the 'nice to do' category.

GetMiterLimit	Returns current miter-join length
SetMiterLimit	Sets miter-join length
StrokeAndFillPath	Closes, strokes and fills a path
StrokePath	Renders a path with the current pen
WidenPath	Sets the current path to pen width
ExtCreatePen	Creates a logical pen

BOOL AbortPath(*hdc*).

HDC *hdc*;

The AbortPath function aborts and discards any paths in the DC. If there is an active path in the DC started by the BeginPath function, the path is ended and discarded. If there is an inactive path in the DC, started by BeginPath and ended by EndPath, the path is discarded.

Parameter	**Description**
hdc	Identifies the device context where paths will be discarded.

Returns

The return value is TRUE if the function was successful, or FALSE if an error occurred. To obtain extended error information, use the GetLastError function.

6.3.2.1. BeginPath

BOOL BeginPath(hdc);

HDC *hdc*;

The **BeginPath** function starts a path bracket; subsequent calls to drawing functions define the shape and size of a path. **BeginPath** initializes the path associated with the DC, discarding the previous path, if any. All subsequent drawing functions called for that DC, up to the next call to EndPath, apply to the new path.

Parameter	**Description**
hdc	Identifies the device context.

Returns

The return value is TRUE if the call succeeded, otherwise FALSE.

Comments

The following drawing functions will add to the path (Note that this list is still under discussion):

CloseFigure
Ellipse
LineTo
MoveToEx
PolyBezier
PolyBezierTo
Polygon
Polyline
PolylineTo
PolyPolygon
PolyPolyline
Rectangle
RoundRect

6.3.2.2. CloseFigure

BOOL CloseFigure(*hdc*)

HDC *hdc*;

The **CloseFigure** function closes an open figure in a path. It should only be called when there is an active path associated with the DC, started by a BeginPath call.

A figure in a path is open unless it is explicitly closed using **CloseFigure**. A figure can be open even if the current point and the starting point of the figure are the same.

This function closes the figure by drawing a line from the current position to the first point of the figure (usually, the point specified by the most recent MoveTo call), and connects the lines using the line join style. If a figure is closed using LineTo instead of CloseFigure, end caps will be used to create the corner instead of a join.

Any line or curve added to the path after **CloseFigure** starts a new figure.

Parameter	**Description**
hdc	Identifies the DC.

Returns

The return value is TRUE if the call succeeded, otherwise FALSE.

Comments

CloseFigure can only be used in a path bracket, started by calling BeginPath.

6.3.2.3. EndPath

BOOL EndPath(*hdc*)

HDC *hdc*;

The **EndPath** function ends a path bracket; that is, it ends the sequence of functions defining the size and shape of the path started by BeginPath.

Parameter	**Description**
hdc	Identifies the DC.

Returns

The return value is TRUE if the call succeeded, otherwise FALSE.

Errors

Use the GetLastError function to retrieve the error value, which may be one of the following:

ERROR_CAN_NOT_COMPLETE
ERROR_INVALID_PARAMETER

6.3.2.4. FillPath

BOOL FillPath(*hdc*)

HDC *hdc*;

The FillPath function closes any open figures in the current path and fills the result using the current brush and polygon filling mode. See SetPolyFillMode for a description of the filling modes.

Parameter	Description
hdc	Identifies the device context.

Returns

The return value is TRUE if the call succeeded, otherwise FALSE.

Comments

The DC must have an inactive path bracket, started by calling BeginPath and ended by calling EndPath. **FillPath** discards the path.

Errors

Use the GetLastError function to retrieve the error value, which may be one of the following:

ERROR_CAN_NOT_COMPLETE
ERROR_INVALID_PARAMETER
ERROR_NOT_ENOUGH_MEMORY

6.3.2.5. FlattenPath

BOOL FlattenPath(*hdc*)

HDC *hdc*;

The **FlattenPath** function transforms the specified path, turning all portions of the path which consist of curves into sequences of lines.

Parameter	**Description**
hdc	Identifies the device context of the path.

Returns

The return value is TRUE if the call succeeded, otherwise FALSE.

Errors

Use the GetLastError function to retrieve the error value, which may be one of the following:

ERROR_CAN_NOT_COMPLETE
ERROR_INVALID_PARAMETER
ERROR_NOT_ENOUGH_MEMORY

Comments

The DC must have an inactive path bracket, started by calling BeginPath and ended by calling EndPath.

6.3.2.6. GetPath

int **GetPath**(*hdc, lpPoints, lpTypes, nSize*)

HDC *hdc*;
LPPOINT *lpPoints*;
LPBYTE *lpTypes*;
int *nSize*;

The **GetPath** function gets all the lines and curves in a path.

Parameter	**Description**
hdc	Identifies the device context of the path.
lpPoints	Points to an array of POINTs in which the path vertices will be placed.
lpTypes	Points to an array of BYTEs in which the vertex types will be placed. Values will be one of the following:

Type	**Meaning**
PT_MOVETO	Specifies this points starts a disjoint figure.
PT_LINETO	Specifies that a line is to be drawn from the previous point in the path to this point.
PT_BEZIERTO	Specifies that this point is a control point or end-point for a Bezier curve. PT_BEZIERTO types always occur in sets of three. The point in the path immediately preceding them defines the start-point for the Bezier curve. The first two PT_BEZIERTO points are the control points, and the third PT_BEZIERTO point is the end-point.

A PT_LINETO or PT_BEZIERTO type may be combined with the following flag (using the bit-wise operator OR) to indicate that the corresponding point is the last point in a figure and the figure should be closed:

Type	Meaning
PT_CLOSEFIGURE	This flag specifies that the figure is automatically closed after the PT_LINETO or PT_BEZIERTO for this point is done. A line is to be drawn from this point to the previous PT_MOVETO point. This flag is combined with the PT_LINETO type for a line, or with the PT_BEZIERTO type of the end-point for a Bezier curve, using the bit-wise operator OR.

nSize	Specifies the total number of POINTs that may be placed in the *lpPoints* array. This must be the same as the number of BYTEs that may be placed in *lpTypes*. If *nSize* is zero, **GetPath** returns the total number of points in the path, and nothing is written to the buffers.

Returns

If *nSize* is non-zero, **GetPath** returns the number of points enumerated. If *nSize* is zero, it returns the total number of points in the path. The function returns -1 if *nSize* is not zero and is less than the number of points in the path.

Comments

The DC must have an inactive path bracket, started by calling BeginPath and ended by calling EndPath. **GetPath** does not affect t path.

The points of the path are returned in logical coordinates. Points are stored in the path in device coordinates, so **GetPath** transforms the points from device coordinates to logical coordinates using the inverse of the current transform.

FlattenPath may be called before **GetPath** so that only line segments will be enumerated.

Errors

Use the **GetLastError** function to retrieve the error value, which may be one of the following:

ERROR_CAN_NOT_COMPLETE
ERROR_INVALID_PARAMETER
ERROR_BUFFER_OVERFLOW

6.3.2.7. PathToRegion

HRGN PathToRegion(*hdc*)

HDC *hdc*;

The **PathToRegion** function creates a region from a path.

Parameter	**Description**
hdc	Identifies the device context of the path.

Returns

Returns the handle to the resulting region if the call succeeds, 0 otherwise.

Comments

The DC must have an inactive path bracket, started by calling BeginPath and ended by calling EndPath. **PathToRegion** discards the path.

The current SetPolyFillMode value is used to create the region.

Errors

Use the GetLastError function to retrieve the error value, which may be one of the following:

ERROR_CAN_NOT_COMPLETE
ERROR_INVALID_PARAMETER
ERROR_NOT_ENOUGH_MEMORY

6.3.2.8. SelectClipPath

BOOL SelectClipPath(*hdc, iMode*)

HDC *hdc*;
int *iMode*;

The SelectClipPath function selects the current path as a clip region for the specified DC.

Parameter	**Description**
hdc	Identifies the device context of the path.
iMode	Specifies the way to use the path. The following values are allowed:

Type	**Meaning**
RGN_AND	The new clip region includes the overlapping areas of the current clip region and the current path (intersection).
RGN_COPY	The new clip region is the current path.
RGN_DIFF	The new clip region includes the areas of the current clip region with those of the current path excluded.
RGN_OR	The new clip region includes the combined areas of the current clip region and the current path (union).
RGN_XOR	The new clip region includes the combined areas of the current clip region and the current path but without the overlapping areas.

Returns

The return value is TRUE if the call succeeded, otherwise FALSE.

Comments

The DC must have an inactive path bracket, started by calling BeginPath and ended by calling EndPath. SelectClipPath discards the path.

Errors

Use the GetLastError function to retrieve the error value, which may be one of the following:

ERROR_CAN_NOT_COMPLETE
ERROR_INVALID_PARAMETER
ERROR_NOT_ENOUGH_MEMORY

6.3.3. DIB Driver

In Windows 3.0 we added device independent bitmaps (DIBs). DIBs support a variety of well defined color packed pixel formats. Because of their color support and portability DIBs are a tool used to create and manipulate color images independent of the target device. Image processing application make use of DIBs today.

The original goal of a DIB driver was to allow applications and GDI to directly modify the same bitmap image. However a much more important goal is to produce the best possible display drivers for 8, 16, 24, and 32 bit formats. The DIB formats match the hardware formats. In essence the display would be treated as a DIB by the DIB driver. **We are out to make the best and fastest dumb frame buffer.** The DIB actually supports a set of helper functions to device drivers. Those functions range from memory device support to a mini driver engine for frame buffer display devices. These helper functions provided by the DIB driver will simplify the complexity of future device drivers, in particular drivers for blitter or frame buffer based displays. There is also the benefit that printing will be more robust and predictable. Most printer drivers rely on the screen driver to do the rasterization, which means that the quality of the printed page depends on the display driver. By retargetting the printer drivers to the DIB driver reliability goes up.

A DIB driver for Windows needs to support 1, 4, 8, 16, 24, and 32 bit per pixel formats.

DIB DCs will be different from traditional memory DCs in the sense that there will be no restrictions on the source and destination of BitBlt.

We will be using 32 bit code where ever appropriate.

There is a complete spec and schedule for the DIB driver under separate cover.

The use of the DIB driver by display drivers is discussed in the display driver spec by keithla.

The three functions added are

CreateDibSection	Creates a DIB that both the application and GDI can write to.
SetDIBColorTable	Sets the color table of a DIB.
SetDIBColorTable	Gets the color table of a DIB.

6.3.3.1. CreateDIBSection

HBITMAP CreateDIBSection(*hdc, lpInfo, fInit, iUsage, lppBits*)

```
HDC hdc;                     /* handle of device context                    */
LPBITMAPINFO lpInfo;         /* address of bitmap data                      */
DWORD fInit;                 /* scanline-ordering flag                      */
DWORD iUsage;                /* color-format flag                           */
LPBYTE *lppBits;             /* address of buffer that receives bitmap data */
```

The **CreateDIBSection** function copies color data for a number of scanlines into the given buffer.

Parameter	**Description**
hdc	Identifies a device context that contains the logical palette that was used to initialize the bitmap's colors.
lpInfo	Points to a BITMAPINFO structure that describes the desired size and format of the bitmap. in addition, this structure contains a logical palette.
fInit	Specifies whether the ordering of scanlines in the bitmap. This argument can be either one of the following constants:

Constant	**Meaning**
BMF_TOPDOWN	The first scanline in the buffer at which lppBits points is the top scanline.
BMF_DIB	The first scanline in the buffer at which lppBits points is the bottom scanline.

iUsage	Specifies whether the bmiColors field of BITMAPINFO structure contains explicit RGB values or indices into the currently realized logical palette. This argument can be one of the following constants:

Constant	**Meaning**
DIB_PAL_COLORS	The BITMAPINFO structure contains an array of 16-bit indices into a logical palette.
DIB_RGB_COLORS	The BITMAPINFO structure contains an array of literal RGB values.

lppBits	Points to a buffer that receives the bitmap bits.

Returns

The return value is a handle that identifies the bitmap if the function is successful. Otherwise, it is NULL.

Comments

The bitmap bits which are stored in the buffer at which *lppBits* points will be aligned on DWORD boundaries.

HIGHLY CONFIDENTIAL

CMS 00013765

MX 2120471
CONFIDENTIAL

RBC 04166

6.3.3.2. SetDIBColorTable

int SetDIBColorTable (*hdc, iStartIndex, iNumIndices, lpRGBQuadTable*)

```
HDC hdc;                              /* handle of device context      */
UINT iStartIndex;                     /* start of quads to change      */
UINT iNumIndices;                     /* number of quads to change     */
LPRGBQUAD lpRGBQuadTable;             /* address of RGB quads          */
```

The **SetDIBColorTable** function sets the color table in a DIB selected into a DC.

Parameter	**Description**
hdc	Identifies a device context that contains the DIB.
iStartIndex	Index at which the set or get starts.
iNumIndices	Number of indices to set or get.
lpRGBQuadTable	Table of RGB quads.

Returns

The return value is the number of colors set if the function is successful. Otherwise, it is NULL.

Comments

The value StartIndex+iNumIndices can exceed the current number of colors or the maximum for the format. This case may extend the total number of colors in the DIB, up to the maximum value.

Errors

Use the GetLastError function to retrieve the error value, which may be one of the following:

- hBitmap is not a valid DIB Bitmap.
- lpRGBQuadTable is too small to hold iNumIndices number of colors.
- iStartIndex is greater than the current number of colors in the DIB (biClrUsed if non zero, or the max for the DIB format).
- DIBs which don't have a color table (>8bpp).

6.3.3.3. GetDIBColorTable

int **GetDIBColorTable** (*hdc, iStartIndex, iNumIndices, lpRGBQuadTable*)

```
HDC hdc;                          /* handle of device context     */
UINT iStartIndex;                 /* start of quads to change     */
UINT iNumIndices;                 /* number of quads to change    */
LPRGBQUAD lpRGBQuadTable;         /* address of RGB quads         */
```

The **SetDIBColorTable** function gets the color table in a DIB selected into a DC.

Parameter	Description
hdc	Identifies a device context that contains the DIB.
iStartIndex	Index at which the set or get starts.
iNumIndices	Number of indices to set or get.
lpRGBQuadTable	Table of RGB quads.

Returns

The return value is the number of colors retrieved if the function is successful. Otherwise, it is NULL.

Comments

The value StartIndex+iNumIndices can exceed the current number of colors or the maximum for the format. The return value is Min (iNumIndices, Actual number of colors - iStartIndex + 1).

Errors

Use the GetLastError function to retrieve the error value, which may be one of the following:

- hBitmap is not a valid DIB Bitmap.
- lpRGBQuadTable is too small to hold iNumIndices number of colors.
- iStartIndex is greater than the current number of colors in the DIB (biClrUsed if non zero, or the max for the DIB format).
- DIBs which don't have a color table (>8bpp).

6.3.4. Device Independent Color

Device independent color (DIC) also has a separate spec, we will only discuss it briefly here.

The concept of DIC is very simple, that images, composed of pixels whose color is communicated through RGB triplets, should appear the same regardless of the target device. It should look the same on the screen as it does on ink-jet as it does on thermal wax, as it does on ink sublimation, as it does on film.

There are four minimum requirements. 1) Definition of a logical color space. 2) Color aware output devices. 3) Color communication between application and device. 4) Ability to for applications to do color separations.

Device independence normally implies that any particular device will not be used to its fullest capabilities or the output will not be the best possible on that device. In the case of color if an application only wants device independence then it has no real control over solid device colors. How do we support applications that wish to support device colors in charts and stuff? Well, the 24 bit RGB is packed into a 32 bit quantity. We're already using some of those eight bits for palette issues, let's just grab one more for this, say 0x04. If the bit is set then the RGB is to interpreted as a device color coordinates. This means that it specifies a solid color. Think of it this way, a device color coordinate triplet means that the driver has no control over how the color is produced. In the case of a brush, if the device were to dither it would be acting against the wishes of the application. (But through CreateDIBBrush( ) an application can specific an exact dither.) If the device cannot satisfy the request the device can fail the call. For example most color printers have only two values for each ink, either on or off. If an application were to call SetPixel (hDC, X, Y, (0x04, 127, 127, 0)) to set a shade of yellow the device should fail the call. It is up to the application to read the device characterization file to determine what the device can do.

Added color definitions:

```
typedef struct tagCIEXYZ
{
   DWORD      ciexyzX;
   DWORD      ciexyzY;
   DWORD      ciexyzZ;
} CIEXYZ;
typedef CIEXYZ FAR* LPCIEXYZ;

typedef struct tagCIEXYZTRIPLE
{
   CIEXYZ      ciexyzRed;
   CIEXYZ      ciexyzGreen;
   CIEXYZ      ciexyzBlue;
} CIEXYZ;
typedef CIEXYZTRIPLE FAR* LPCIEXYZTRIPLET;
```

The functions added are:

EnableDIC	allows app to force color matching
EnumNamedDeviceColors	enumerates named device colors
SetColorSpace	defines the endpoints of the logical RGB space in the CIEXYZ space
GetColorSpace	gets the endpoints of the logical RGB space in the CIEXYZ space
StartSeparation	allows applications to do color separations
SetOverprint	determines how separations are to be printed
GetCharacterizationTable	retrieves a devices color characteristics (Or pass name of file?)
SetCharacterizationTable	sets a devices color characteristics (Or perhaps pass name of file?)

6.3.4.1. EnableDIC

BOOL EnableDIC(*hDC*, *fEnableDIC*)

```
HDC hdc;              /* handle of device context              */
BOOL fEnableDIC;      /* turns image color matching on and off  */
```

The **EnableDIC** function causes image color matching to be performed on the given DC.

Parameter	**Description**
hdc	Identifies a device context that contains the logical palette that was used to initialize the bitmap's colors.
fEnableDIC	Turns image color matching on and off. TRUE turns it on, FALSE turns it off.

Returns

The return value is true if the function is successful. Otherwise, it is NULL.

Comments

EnableDIC overrides the default value set by the user using the Control panel's Color section.

6.3.4.2. EnumNamedDeviceColors

int EnumNamedDeviceColors(*hdc, lpszNamedColor, lpEnumColorFunc, lParam*)

```
HDC hdc;                            /* device-context handle              */
LPSZSTR lpszNamedColor;             /* address of family-name string      */
ENUMCOLORSPROC lpEnumColorFunc;     /* address of callback function       */
LPARAM lParam;                      /* address of application-supplied data */
```

The **EnumNamedDeviceColors** function either queries for a specific name or enumerates the named colors a device supports.

Parameter	**Description**
hdc	Identifies the device context.
lpszNamedColor	Points to a null-terminated string that specifies the name of the desired color. If not NULL then that color is enumerated first if it exists. If lpszFamily is NULL, EnumNamedDeviceColors enumerates all named colors the device supports in no particular order.
lpEnumColorFunc	Specifies the procedure-instance address of the application-defined callback function. See the description of the EnumColorsProc function.
lParam	Points to application-supplied data. The data is passed to the callback function along with the color information.

Returns

The return value specifies the last value returned by the callback function. Its meaning is implementation specific.

6.3.4.3. EnumColorsProc

int CALLBACK EnumColorsProc(*lpszColorName, lpCIEXYZ, lpData*)

```
lpszSTRING lpszColorName;   /* pointer to name of color    */
lpCIEXYZ lpCIEXYZ;          /* address of coefficients     */
LPARAM lpData;              /* application-supplied data   */
```

The EnumFontsProc function is an application defined callback function that processes font data from the EnumFonts function.

Parameter	**Description**
lpszColorName	Points to a null terminated color name.
lpCIEXYZ	Points to the CIEXYZ definition of the color.
lpData	Points to the application-supplied data passed by EnumNamedDeviceColors.

Returns

This function must return a nonzero value to continue enumeration; to stop enumeration, it must return zero.

6.3.4.4. SetColorSpace

BOOL SetColorSpace(*hdc, lpciexyzRGB, ColorSpace*)

```
HDC hdc;                          /* handle of device context              */
LPCIEXYZTRIPLET lpciexyzRGB;      /* pointer to rgb definition in ciexyz space */
UINT ColorSpace                   /* color space to set                    */
```

The **SetColorSpace** defines the endpoints of the logical RGB space in CIEXYZ space.

Parameter	**Description**
hdc	Identifies a device context that is to have the given color space definition.
lpciexyzRGB	RGB definition in CIEXYZ space.
ColorSpace	Determines what color space to set. It can have the following meanings:

Constant	**Meaning**
DEVICE_RGB	RGBQuads defined in API's are to be interpreted as device RGB's. In effect this turns color matching off.
DEVICE_CMYK	RGBQuads defined in API's are to be interpreted as device CMYK. No translation of these values are allowed. The CIEXYZTriplet is ignored in this case.
CALIBRATED_RGB	The given CIEXYZTriplet defines the endpoints of the RGB's used in all API's.

Returns

The return value is TRUE if the function is successful. Otherwise, it is NULL.

6.3.4.5. GetColorSpace

BOOL GetColorSpace(*hdc, lpciexyzRGB, ColorSpace*)

```
HDC hdc;                           /* handle of device context                      */
LPCIEXYZTRIPLET lpciexyzRGB;       /* pointer to put rgb definition in ciexyz space */
UINT ColorSpace                    /* color space defined                           */
```

The GetColorSpace retrieves the current endpoints of the logical RGB space in CIEXYZ space.

Parameter	Description
hdc	Identifies a device context that is to have the given color space definition.
lpciexyzRGB	RGB definition in CIEXYZ space.
ColorSpace	Determines what color space is set. It can have the following meanings:

Constant	Meaning
DEVICE_RGB	RGBQuads defined in API's are to be interpreted as device RGB's. In effect this turns color matching off.
DEVICE_CMYK	RGBQuads defined in API's are to interpreted as device CMYK. No translation of these values are allowed. The CIEXYZTriplet is ignored in this case.
CALIBRATED_RGB	The given CIEXYZTriplet defines the endpoints of the RGB's used in all API's.

Returns

The return value is TRUE if the function is successful. Otherwise, it is NULL.

6.3.4.6. StartSeparation

BOOL StartSeparation(*hdc, lpszColorName, Coef1, Coef2, Coef3, Coef4, ColorSpace*)

```
HDC hdc;                          /* handle of device context                  */
LPSZSTRING lpszColorName;         /* pointer to zero terminated name of color  */
ULONG Coef1;                      /* first coefficient                         */
ULONG Coef2;                      /* second coefficient                        */
ULONG Coef3;                      /* third coefficient                         */
ULONG Coef4;                      /* fourth coefficient                        */
UINT ColorSpace                   /* color space                               */
```

The **StartSeparation** function causes a monochrome image to be accumulated and to be displayed with the given color.

Parameter	**Description**
hdc	Identifies the device context.
lpszColorName	Pointer to the name of the requested color.
Coef1	Used to define the requested color in a device independent way.
Coef2	
Coef3	
Coef4	
ColorSpace	Determines the color space the requested color is defined against. The coefficients above define the requested color in the color space. It can have the following meanings:

Constant	**Meaning**
DEVICE_RGB	Color = Coef1*R + Coef2*B + Coef3*G
DEVICE_CMYK	Color = Coef1*C + Coef2*M + Coef3*Y +Coef4*K
CALIBRATED_RGB	Color = Coef1*R + Coef2*B + Coef3*G The RBG is the DC's logical RGB.

Returns

The return value is TRUE if the function is successful. Otherwise, it is NULL.

Comments

If the device does not support the named color, then it uses the color space definition to match the color wanted. The image is accumulated until the next **StartSeparation** or until EndPage. **(Should we define an EndSeparation for symmetry?)** When EndPage is called the separations printed. Overprinting or transparency is supported by the SetOverPrinting function.

6.3.4.7. SetOverPrinting

```
BOOL SetOverPrinting(hdc, fOver)

HDC hdc;                    /* handle of device context    */
BOOL fOver;                 /* flag for overprinting       */
```

The SetOverPrinting function determine how separations are to be put together.

Parameter	**Description**
hdc	Identifies the device context.
fOver	Determines overprint mode. If TRUE then separations print over each other. If FALSE then the top separations cause the images below them to be erased for those areas of the separation that are not opaque.

Returns

Returns the value of the current value.

Comments

The default value is FALSE.

6.3.4.8. GetCharacterizationTable

BOOL GetCharacterizationTable(*hdc*, *lpTable*)

```
HDC hdc;               /* handle of device context    */
LPVOID lpTable;        /* pointer to table to fill    */
```

The **GetCharacterization** function retrieves the characterization information for the device.

Parameter	**Description**
hdc	Identifies the device context.
lpTable	Pointer to table to fill with device's characterization.

Returns

The return value is TRUE if the function is successful. Otherwise, it is NULL.

Comments

THE FORMAT OF THE CHARACTERIZATION INFORMATION IS NOT YET DEFINED. Applications or DLL's may want to do their own color and gamut matching. (Or perhaps pass name of file?)

6.3.4.9. SetCharacterizationTable

BOOL SetCharacterizationTable(*hdc*, *lpTable*)

```
HDC hdc;               /* handle of device context    */
LPVOID lpTable;        /* pointer to table to copy    */
```

The **SetCharacterizationTable** sets the characterization information for the device.

Parameter	**Description**
hdc	Identifies the device context.
lpTable	Pointer to the table to copy.

Returns

The return value is TRUE if the function is successful. Otherwise, it is NULL.

Comments

THE FORMAT OF THE CHARACTERIZATION INFORMATION IS NOT YET DEFINED. Users will have the ability to recharacterize their devices. Devices may drift, or new printing paper may be introduced, or the dithering scheme may be upgraded, or the phosphors get old, etc. We expect third parties to to supply tools to do this. (Or perhaps pass name of file?)

6.3.5. Color Cursors

Geez, OS/2, NT, and all of our competitors support color cursors. To see what this means just move icons around and watch them turn monochrome. We have lots of code we can grab from OS/2 and NT to do this.

6.3.6. Metafiles

What are the problems with metafiles? And why should we fix them? Let us start with the problems first.

The basic problem is that metafiles are not really device independent. Some metafile records use device coordinates instead of logical coordinates, non-linear text cannot be properly scaled, no idea as to the intended size of the image represented by the metafile, etc. Windows NT has addressed these problems.

The 7 limitations in 16 bit metafiles which NT fixes are:

1. Device dependent units - 16 bit metafile is device dependent in MM_TEXT mode.
2. No bounds information. - Have no idea how big the picture is before it is played.
3. Not scalable. - metafile transform overwrites playback transform.
4. Clipping is device dependent.
5. Different programming interface (some fixed in 3.1)
6. Font selections - NT records panose number as well as facenames.
7. No way to determine the set of RGB's important in the metafile.

The second problem is the addition of new records to metafiles. As we add API's to GDI, like those in the previous section, we must be able to record these functions in metafiles so new metafile records are created. But if a metafile with new records gets played on Windows 3.1 then these records will be ignored. This problem is not a metafile problem per se, but it is a real problem none the less. Either the system and/or applications are going to have to construct metafiles without new records to be played on 3.1.

In order to address both of the above problems it has been proposed that Chicago support the NT enhanced metafiles.

The functions added are:

CloseEnhMetaFile	Closes an enhanced metafile DC
CopyEnhMetaFile	Copies an enhanced metafile
CreateEnhMetaFile	Creates an enhanced metafile DC
DeleteEnhMetaFile	Invalidates enhanced metafile handle
EnumEnhMetaFile	Returns GDI calls within an enhanced metafile
EnumMetaFileProc	Processes metafile data
GdiComment	Adds a comment to an enhanced metafile
GetEnhMetaFile	Creates an enhanced metafile
GetEnhMetaFileBits	Copies enhanced metafile bits to a buffer
GetEnhMetaFileDescription	Returns creator and title for enhanced metafile
GetEnhMetaFileHeader	Returns enhanced metafile header
GetEnhMetaFilePaletteEntries	Returns enhanced metafile palette entries
GetWinMetaFileBits	Retrieves metafile contents in Windows format
PlayEnhMetaFile	Plays an enhanced metafile to a DC
PlayEnhMetaFileRecord	Plays an enhanced metafile record
SetEnhMetaFileBits	Creates a memory-based enhanced metafile from data
SetWinMetaFileBits	Creates enhanced metafile from Windows metafile data

6.3.6.1. CloseEnhMetaFile

HENHMETAFILE CloseEnhMetaFile(*hdc*)

HDC *hdc*; /* handle of an enhanced-metafile device context */

The **CloseEnhMetaFile** function closes an enhanced-metafile device context and returns a handle that identifies an enhanced metafile.

Parameter	Description
hdc	Identifies an enhanced-metafile device context.

Returns

The return value is a handle that identifies an enhanced metafile if the function is successful. Otherwise, it is NULL.

Comments

An application can use the enhanced metafile handle that was returned by this function to perform the following tasks:

- Display a picture that is stored in an enhanced metafile
- Create copies of the enhanced metafile
- Enumerate, edit, or copy individual records in the enhanced metafile
- Retrieve an optional description of the metafile contents from the enhanced metafile header
- Retrieve a copy of the enhanced-metafile header
- Retrieve a binary copy of the enhanced metafile
- Enumerate the colors in the optional color-palette

When the application no longer needs the enhanced metafile handle, it should call the DeleteEnhMetaFile function.

6.3.6.2. CopyEnhMetaFile

```
HENHMETAFILE CopyEnhMetaFile(hemfSrc, lpszFile)

HENHMETAFILE hemfSrc;      /* handle of an enhanced metafile  */
LPTSTR lpszFile;           /* address of a filename string    */
```

The **CopyEnhMetaFile** function copies the contents of an enhanced metafile to the specified file.

Parameter	**Description**
hemfSrc	Identifies the source enhanced metafile.
lpszFile	Points to the name of the file that is to receive the enhanced metafile. If NULL, the metafile is copied to memory

Returns

The return value identifies a copy of the enhanced metafile if the function is successful. Otherwise, it is NULL. Use the GetLastError function to obtain extended error information.

Comments

If *lpszFile* is NULL, the metafile is copied to memory; otherwise it is copied to disk. Applications can use metafiles stored in memory for temporary operations. An application should delete an enhanced metafile once it is no longer needed by calling the DeleteEnhMetaFile function.

6.3.6.3. CreateEnhMetaFile

HDC CreateEnhMetaFile(*hdcRef, lpFilename, lpRect, lpDescription*)

```
HDC hdcRef;                  /* handle of a reference device-context       */
LPTSTR lpFilename;           /* address of a filename string               */
LPRECT lpRect;               /* address of a bounding rectangle            */
LPTSTR lpDescription;        /* address of an optional description-string  */
```

The **CreateEnhMetaFile** function creates an enhanced-metafile device context that can be used to store a device-independent picture.

Parameter	**Description**
hdcRef	Identifies a reference device for the enhanced metafile.
lpFilename	Points to a filename in which the enhanced metafile is stored.
lpRect	Points to a RECT structure that specifies the dimensions (in .01 mm units) of the picture that will be stored in the enhanced metafile.
lpDescription	Points to a string that specifies the name of the application that created the picture as well as the picture's title.

Returns

The return value identifies the device context for the enhanced metafile if the function is successful. Otherwise, it is NULL.

Comments

Windows uses the reference device identified by *hdcRef* to record the resolution and color format of the device on which a picture originally appeared.

If *lpFilename* is NULL, the enhanced metafile is stored temporarily in memory.

The *left* and *top* members of the RECT structure at which *lpRect* points must be less than the *right* and *bottom* members respectively. Points along the edges of the rectangle are included in the picture. If *lpRect* is NULL, GDI computes the dimensions of the smallest rectangle that will surround the picture drawn by the application.

The string at which *lpDescription* points must contain two null characters between the application and the picture name. For example: "XYZ Graphics Editor\0\0Bald Eagle\0" where '\0' represents the null character). If *lpDescription* is NULL, there is no corresponding entry in the enhanced-metafile header.

Applications use the device context (DC) created by this function to store a graphics picture in an enhanced metafile. The handle identifying this DC can be passed to any GDI drawing function.

Once an application stores a picture in an enhanced metafile, it can display the picture on any output device (by calling the PlayEnhMetaFile function.) When displaying the picture, Windows uses the rectangle at which the *lpRect* structure points and the resolution data (from the reference device) to position and scale the picture.

The device context returned by this function contains the same default attributes associated with any new DC.

Applications must use the GetWinMetaFileBits function to convert an enhanced metafile to the older Windows-metafile format.

It is recommended that the filename for the enhanced metafile use the .EMF extension.

6.3.6.4. DeleteEnhMetaFile

BOOL DeleteEnhMetaFile(*hemf*)

HENHMETAFILE *hemf*; /* handle of an enhanced metafile */

The **DeleteEnhMetaFile** function deletes a metafile or a metafile handle.

Parameter	**Description**
hemf	Identifies an enhanced metafile.

Returns

The return value is TRUE if the metafile has been invalidated. Otherwise, it is FALSE.

Comments

If *hemf* identifies an enhanced metafile that is stored in memory, this function deletes the metafile. If hemf identifies a metafile that is stored on disk, this function deletes the metafile handle hemf but the actual metafile is not destroyed. (An application can retrieve the file by calling the GetEnhMetaFile function.)

6.3.6.5. EnumEnhMetaFile

BOOL EnumEnhMetaFile(*hdc, hemf, lpEnhMetaFunc, lpData, lpRect*)

```
HDC hdc;                      /* handle of a device context              */
HENHMETAFILE hemf;            /* handle of an enhanced metafile          */
PROC lpEnhMetaFunc;           /* address of a callback function          */
LPVOID lpData;                /* address of callback-function data       */
LPRECT lpRect;                /* address of a picture's bounding rectangle */
```

The **EnumEnhMetaFile** function enumerates the records within an enhanced metafile by retrieving each record and passing it to the specified callback function.

Parameter	**Description**
hdc	Identifies a device context. This handle is passed to the callback function.
hemf	Identifies the enhanced metafile.
lpEnhMetaFunc	Points to the application-supplied callback function.
lpData	Points to optional callback-function data.
lpRect	Points to a RECT structure that specifies the coordinates of the picture's upper-left and lower-right corners. The dimensions of this rectangle are specified in logical units.

Returns

The return value is TRUE if the callback function enumerates all the records in the enhanced metafile. Otherwise, it is FALSE.

Comments

Points along the edge of the rectangle at which *lpRect* points are included in the picture. If *hdc* is NULL, Windows ignores *lpRect*.

If the callback function calls the PlayEnhMetaFile function, *hdc* must identify a valid device context. (Windows uses the device context's transformation and mapping mode to transform the picture that is displayed by the PlayEnhMetaFile function. If *hdc* is NULL, the picture will not be transformed and may appear distorted.)

Applications can use the EnumEnhMetaFile function to embed one enhanced metafile within another.

6.3.6.6. EnumMetaFileProc

```
int CALLBACK EnumMetaFileProc(hDC, lpHTable, lpMFR, nObj, lpClientData)

HDC hDC;                          /* handle of device context      */
HANDLETABLE FAR *lpHTable;        /* address of metafile handle-table */
METARECORD FAR *lpMFR;            /* address of metafile record    */
int nObj;                         /* count of objects              */
LPARAM lpClientData;              /* address of optional data */
```

The **EnumMetaFileProc** function is an application-defined callback function that processes metafile records. This function is called by the EnumMetaFile function.

Parameter	**Description**
hDC	Identifies a metafile device-context.
lpHTable	Points to a table of handles associated with the graphic objects (pens, brushes, and so on) in the metafile.
lpMFR	Points to one of the records in the metafile.
nObj	Specifies the number of objects with associated handles in the handle table.
lpClientData	Points to any application-supplied data.

Returns

This function must return a nonzero value to continue enumeration; to stop enumeration, it must return zero.

Comments

An application must register the function by passing its address to the EnumMetaFile function.

EnumMetaFileProc is a placeholder for the application-supplied function name. The actual name must be exported by including it in an EXPORTS statement in the application's module-definition file.

6.3.6.7. GdiComment

```
BOOL GdiComment(hdc, nSize, lpData)

HDC hdc;              /* handle of a device context   */
UINT nSize;           /* size of text buffer          */
LPBYTE lpData;        /* address of text buffer       */
```

The GdiComment function copies the comment from a buffer into the enhanced metafile that has been specified.

Parameter	Description
hdc	Identifies the device context.
nSize	Specifies the length of the comment buffer in bytes.
lpData	Points to the buffer that contains the comment.

Returns

The return value is TRUE if the function is successful. Otherwise, it is FALSE.

Comments

A comment may include any private information. Examples are the the source of the picture and the date upon which it was created. (A comment should begin with an application signature followed by the data.)

Comments should not contain application-specific data nor should they contain position-specific data. (Position-specific data is data specifying the location of a record–it should not be included since one metafile may be embedded within another metafile.)

6.3.6.8. GetEnhMetaFile

HENHMETAFILE GetEnhMetaFile(*lpszMetaFile*)

LPTSTR *lpszMetaFile*; /* address of metafile name */

The **GetEnhMetaFile** function creates a handle that identifies the enhanced metafile which is stored in the specified file.

Parameter	**Description**
lpszMetaFile	Points to a null-terminated filename that contains the enhanced metafile.

Returns

The return value identifies the enhanced metafile if the function is successful. Otherwise, it is NULL.

Comments

When the application no longer needs an enhanced metafile hand[illegible] it should call the DeleteEnhMetaFile function.

A Windows-format metafile must be converted to the enhanced format (by calling SetWinMetaFileBits) before it can be processed by the GetEnhMetaFile function.

6.3.6.9. GetEnhMetaFileBits

UINT GetEnhMetaFileBits(*hemf*, *cbBuffer*, *lpbBuffer*)

HENHMETAFILE *hemf*;	/* handle of metafile */
UINT *cbBuffer*;	/* size of data buffer */
LPBYTE *lpbBuffer*;	/* address of data buffer */

The **GetEnhMetaFileBits** function retrieves the contents of the specified enhanced metafile and copies them into a buffer.

Parameter	**Description**
hemf	Identifies the enhanced metafile.
cbBuffer	Specifies the size, in bytes, of the buffer reserved for the data. Only this many bytes will be written.
lpbBuffer	Points to the buffer to receive the metafile data. The buffer must be sufficiently large to contain the data. If *lpData* is NULL, the function returns the size necessary to hold the data.

Returns

If the function is successful and *lpData* is NULL, the return value specifies the size of the enhanced metafile in bytes; if the function is successful and *lpData* is a valid pointer, the return value specifies the number of bytes copied. The return value is zero if an error occurs.

Comments

Once the enhanced metafile bits are retrieved, they can be used to create a memory-based metafile by calling the SetEnhMetaFileB: function.

The GetEnhMetaFileBits function does not invalidate the enhanced metafile handle hemf. An application should free the handle by calling the DeleteEnhMetaFile function when it no longer needs the handle.

The metafile contents retrieved in this function is in the enhanced format. To retrieve the metafile contents in the Windows 3 0 format, use the GetWinMetaFileBits function.

6.3.6.10. GetEnhMetaFileDescription

```
UINT GetEnhMetaFileDescription(hemf, cchBuffer, lpszDescription)

HENHMETAFILE hemf;            /* handle of enhanced metafile      */
UINT cchBuffer;               /* size of text buffer in characters */
LPTSTR lpszDescription;       /* address of text buffer           */
```

The **GetEnhMetaFileDescription** function retrieves an optional text description from an enhanced metafile and copies the string to the specified buffer.

Parameter	**Description**
hemf	Identifies the enhanced metafile.
cchBuffer	Specifies the size, in characters, of the buffer at which lpszDescription points. This is the maximum number of characters copied.
lpszDescription	Points to the buffer into which the function copies the optional text description.

Returns

If the optional text description does not exist, the return value is zero. If the text description exists but lpszDescription is NULL, the return value is the length of the string in characters. If the text description exists and lpszDescription is a valid pointer, the return value specifies the number of characters copied into the buffer. Otherwise it is GDI_ERROR.

Comments

The optional text description contains two strings, the first identifies the application used to create the enhanced metafile and the second identifies the picture contained in the metafile. The two strings are separated by two null characters. For example: "XYZ Graphics Editor\0\0Bald Eagle\0" where '\0' represents the NULL character.

6.3.6.11. GetEnhMetaFileHeader

UINT GetEnhMetaFileHeader(*hemf, cbBuffer, lpemh*)

```
HENHMETAFILE hemf;              /* handle of enhanced metafile        */
UINT cbBuffer;                  /* size of buffer in bytes            */
LPENHMETAHEADER lpemh;          /* address of buffer that receives data */
```

The **GetEnhMetaFileHeader** function retrieves the record which contains the header for the specified enhanced metafile.

Parameter	**Description**
hemf	Identifies the enhanced metafile for which the header is retrieved.
cbBuffer	Specifies the number of bytes to be read. Only this many bytes will be written.
lpemh	Points to a ENHMETAHEADER structure that receives the header record. If NULL, the function returns the size of the header record.

Returns

If *lpemh* is NULL, the return value specifies the size of the record which contains the header; if *lpemh* is a valid pointer, the return value specifies the number of bytes copied. Otherwise it is zero.

Comments

An enhanced metafile header contains information such as the metafile's size in bytes, the dimensions of the picture stored in the metafile, the number of records stored in the metafile, the offset to the optional text description, the offset to the optional color palette, the size of the optional color palette, and the resolution of the device upon which the picture was created.

The record that contains the enhanced metafile header is always the first record in the metafile.

6.3.6.12. GetEnhMetaFilePaletteEntries

UINT GetEnhMetaFilePaletteEntries(*hemf*, *cEntries*, *lppe*)

```
HENHMETAFILE hemf;        /* handle of enhanced metafile        */
UINT cEntries;            /* count of palette entries           */
LPPALETTEENTRY lppe;      /* address of PALETTEENTRY array      */
```

The GetEnhMetaFilePaletteEntries function retrieves optional color-palette entries from the specified enhanced metafile.

Parameter	Description
hemf	Identifies the enhanced metafile.
cEntries	Specifies the number of entries to be retrieved from the optional color palette.
lppe	Points to an array of PALETTEENTRY structures that will receive the palette colors. The array must contain at least as many structures as specified by the cEntries parameter.

Returns

If *lppe* is NULL and the enhanced metafile contains an optional color palette, the return value is the number of entries in the enhanced-metafile's color palette; if *lppe* is a valid pointer and the enhanced metafile contains an optional color palette, the return value is the number of entries copied; if the *lppe* is a valid pointer but the metafile does not contain an optional color palette, the return value is zero. The return value is GDI_ERROR if an error occurs.

Comments

An application can store an optional color palette in an enhanced metafile by calling the CreatePalette and SetPaletteEntries functions prior to creating the picture and storing it in the metafile. By doing this, the application can guarantee consistent colors when the picture is displayed on a variety of devices.

Applications which display a picture stored in an enhanced metafile should call the GetEnhMetaFilePaletteEntries function to determine whether the optional palette exists, and, if it does, call the GetEnhMetaFilePaletteEntries function a second time to retrieve those entries. Once the application has retrieved the color palette entries, it should create a logical palette, select it into its device context, and then realize it. These operations are performed by the CreatePalette, SelectPalette, and RealizePalette functions. Once the logical palette has been realized, the application can display the picture with its original colors by calling the PlayEnhMetaFile function.

6.3.6.13. GetWinMetaFileBits

UINT GetWinMetaFileBits(*hemf, cbBuffer, lpbBuffer, fnMapMode, hdcRef*)

```
HENHMETAFILE hemf;     /* handle of the enhanced metafile         */
UINT cbBuffer;         /* buffer size                             */
LPBYTE lpbBuffer;      /* pointer to buffer                       */
INT fnMapMode;         /* mapping-mode identifier                 */
HDC hdcRef;            /* handle of reference device-context      */
```

The **GetWinMetaFileBits** function converts the enhanced-format records from a metafile into the Windows-format records and stores the converted records in the specified buffer.

Parameter	Description
hemf	Identifies the enhanced metafile.
cbBuffer	Specifies the size, in bytes, of the buffer into which the converted records are copied. Only this many bytes will be written.
lpbBuffer	Points to the buffer into which the converted records are copied.
fnMapMode	Specifies the mapping mode that was used to create the picture stored in the enhanced metafile.
hdcRef	Identifies the reference device.

Returns

If the function is successful, the return value is the size of the metafile data in bytes. Otherwise, it is zero.

Comments

This function is used to convert an enhanced metafile into an older Windows-format metafile so that its picture can be displayed in an application that recognizes the older format.

If *lpbBuffer* is NULL, **GetWinMetaFileBits** returns the the number of bytes required to store the converted metafile records.

Windows uses the reference device context to determine the resolution of the original enhanced metafile.

The GetWinMetaFileBits function does not invalidate the enhanced metafile handle hemf. An application should free the handle by calling the DeleteEnhMetaFile function when it no longer needs the handle.

Due to the limitations of Windows 3.x, some information may be lost in the retrieved metafile contents. For example, an original PolyBezier call in the enhanced metafile may be converted into a Polyline call in the Windows metafile since there is no equivalent PolyBezier call in Windows 3.x.

Windows 3.x applications define the viewport origin and extents of a picture that is stored in a Windows-format metafile. As a result, the Windows-format records that are created by this function do not contain any SetViewportOrg and SetViewportExt functions. GetWinMetaFileBits will, however, create Windows-format records for the SetWindowExt and SetMapMode functions.

To create a scalable Windows-format metafile, the *fnMapMode* parameter should specify MM_ANISOTROPIC. The upper left corner of the metafile picture is always mapped to the origin of the reference device.

6.3.6.14. PlayEnhMetaFile

BOOL PlayEnhMetaFile(*hdc, hemf, lpRect*)

```
HDC hdc;                    /* handle of a device context      */
HENHMETAFILE hemf;          /* handle of an enhanced metafile  */
LPRECT lpRect;              /* address of bounding rectangle   */
```

The **PlayEnhMetaFile** function displays the picture stored in the specified enhanced metafile.

Parameter	**Description**
hdc	Identifies the device context for device upon which the picture will appear.
hemf	Identifies the enhanced metafile.
lpRect	Points to a RECT structure that contains the coordinates of the bounding rectangle that is used to display the picture. The coordinates are specified in logical units.

Returns

The return value is TRUE if the function was successful, or FALSE if an error occurred. Use the GetLastError function to obtain extended error information.

Comments

When an application calls the **PlayEnhMetaFile** function, Windows uses the picture frame in the enhanced-metafile header to map he picture onto the rectangle at which *lpRect* points. (This rectangle may be sheared or rotated prior to calling PlayEnhMetaFile.) Points along the edges of the rectangle at which *lpRect* points are included in the picture.

An enhanced metafile picture can be clipped by defining the clipping region in the output device before playing the enhanced metafile.

An enhanced metafile can be embedded in a newly created enhanced metafile by calling PlayEnhMetaFile and playing the source enhanced metafile into the device context for the new enhanced metafile.

The states of the output device context are preserved in this function. Any object created but not deleted in the enhanced metafile is deleted by this function.

To abort this function, an application can call CancelDC from another thread to terminate the operation. In this case, the function will return FALSE.

6.3.6.15. PlayEnhMetaFileRecord

BOOL PlayEnhMetaFileRecord(*hdc, lpHandletable, lpEnhMetaRecord, nHandles*)

```
HDC hdc;                                /* handle of device context          */
LPHANDLETABLE lpHandletable;            /* address of metafile handle-table  */
LPENHMETARECORD lpEnhMetaRecord;        /* address of metafile record        */
UINT nHandles;                          /* count of handles                  */
```

The **PlayEnhMetaFileRecord** function plays an enhanced-metafile record by executing the GDI calls that are identified by the record.

Parameter	**Description**
hdc	Identifies the device context that was passed to EnumEnhMetaFile.
lpHandletable	Points to the object handle-table to be used for the enhanced metafile playback. The first entry in this table contains the enhanced metafile handle.
lpEnhMetaRecord	Points to the enhanced-metafile record to be played.
nHandles	Specifies the number of handles in the handle table.

Returns

The return value is TRUE if the function is successful. Otherwise, it is FALSE.

Comments

This is an enhanced metafile function.

This function should be called by the EnumEnhMetaFileProc function to play a metafile one record at a time.

The *hdc*, *lpHandletable*, and *nHandles* parameters must be exactly those passed to the *EnhMetaFunc* callback procedure by EnumEnhMetaFile.

If the PlayEnhMetaFileRecord function does not recognize a record, it will ignore the record and return TRUE. If the *hdc* parameter refers to a device context for an enhanced metafile, the unrecognized record will be embedded in the enhanced metafile without modification.

6.3.6.16. SetEnhMetaFileBits

HENHMETAFILE SetEnhMetaFileBits(*cbBuffer*, *lpData*)

```
UINT cbBuffer;       /* buffer size                                  */
LPBYTE lpData;       /* buffer that contains enhanced-metafile data  */
```

The **SetEnhMetaFileBits** function creates a memory-based enhanced metafile from the supplied data.

Parameter	**Description**
cbBuffer	Specifies the size, in bytes, of the data provided.
lpData	Points to a buffer that contains enhanced-metafile data.

Returns

The return value identifies a memory-based enhanced metafile if the function is successful. Otherwise, it is NULL.

Comments

It is assumed that the data which *lpData* contains was obtained by calling the GetEnhMetaFileBits function.

When the application no longer needs the enhanced metafile handle, it should call the DeleteEnhMetaFile function.

The SetEnhMetaFileBits function does not accept metafile data in the windows format. To import the Windows-format metafiles, use the SetWinMetaFileBits function.

6.3.6.17. SetWinMetaFileBits

HENHMETAFILE SetWinMetaFileBits(*cbBuffer, lpbBuffer, hdcRef, lpmfp*)

```
UINT cbBuffer;                 /* size of buffer                      */
LPBYTE lpbBuffer;              /* address of buffered metafile data */
HDC hdcRef;                    /* handle of reference device-context  */
LPMETAFILEPICT lpmfp;          /* suggested size of metafile picture  */
```

The SetWinMetaFileBits function converts a metafile from the older Windows format to the new enhanced format and stores the new metafile in memory.

Parameter	Description
cbBuffer	Specifies the size, in bytes, of the buffer that contains the Windows-format metafile.
lpbBuffer	Points to a buffer that contains the Windows 3.x metafile data. It is assumed that the data was previously created using the GetMetaFileBitsEx or GetWinMetaFileBits function.
hdcRef	Identifies a reference device.
lpmfp	Points to a METAFILEPICT structure that contains the suggested size of the metafile picture and the mapping mode that was used when the picture was created.

Returns

The return value identifies a memory-based enhanced metafile if the function is successful. Otherwise, it is NULL. Use the GetLastError function to obtain extended error information.

Comments

Windows uses the reference device's resolution data and the data in the METAFILEPICT structure to scale a picture. If hdcRef is NULL, Windows will use resolution data for the current output device. If lpmfp is NULL, Windows uses the MM_ANISOTROPIC mapping mode to scale the picture so that it fits the entire device surface.

When the application no longer needs the enhanced metafile handle, it should free the handle by calling the DeleteEnhMetaFile function.

The handle returned by this function can be used with other enhanced metafile functions.

If the reference device is not identical to the device in which the metafile is originally created, some GDI functions which use device units may not draw correctly.

6.3.7. TrueType

This consists mostly of bug fixes, performance enhancements, and support for the new and improved 32-bit rasterizer from STAT. There are presently no new API's planned for TrueType.

6.3.7.1. New rasterizer support

There are a few problems with the current rasterizer. Most of these are related to limitations of doing 16 bit math instead of 32-bit math. Some examples: 1) at large point sizes the glyphs are not right due to cummulative errors which are not small; 2) Windows cannot rasterize certain glyphs at certain point sizes. There is also the problem that 'faults' in the font programs bring the rasterizer to its knees and our users must restart Windows to get it working again. The STAT group is working on a 32-bit version that in addition to solving these problems also is doing more performance work, such as rewriting the scan converter to use better algorithms. The interface to the rasterizer must also be replaced. It should be based on the designed interface of the rasterizer; currently there is a layer between Windows and the rasterizer that must be removed. By removing this layer we will be able to take complete advantage of TrueType fonts, for example currently we cannot support optically scaling fonts.

6.3.7.2. Performance enhancements

This is a grab bag of things we know need to get fixed. GetCharABCWidths is too slow to be usable. Booting Windows with lots of TrueType fonts installed is very slow. Reselection of fonts can be slow. (I need to go through all of my old email to see what else there is.)

6.3.8. Reduce GDI local heap limitations

Should be obvious, we should move stuff out of GDI's heap into their own heaps. Regions are on their way out. Physical devices can be moved out. Strings can be moved out. Some objects can be made fixed. Neilk's local memory stuff. Etc.

6.3.9. Robustness

This is a placeholder for object ownership.

6.3.10. Reentrant GDI

Goes without saying. The goal is 'smooth multitasking'. I define this to mean that the USER async input model works, and that printing can be done in the background without affecting the performance of the screen. More details to follow as we figure this stuff out.

6.3.11. postponed bugs

There are about 50 t 100 bugs 3.1 postponed against GDI. I do not know right now how many postponed bugs are against the display dirvers.

6.3.12. General Performance

This is a place holder section.

gradient fills
polyline in driver
clip regions exposed
region generation
color matching
analyze and do something about all of russbl's data
neils seletobject stuff
reduce swap tuning
disjoint polyline
profile profile profile

6.3.13. Masked bitblts

NT supports this. Our ISV's have asked for this. Multimedia has asked for this. This is supported in the MM video drivers to some extent. The Mac has this, it's called CopyMask() in Quickdraw.

6.3.14. Installable decompressors

More to follow, if I have the time, if I don't have the time to fill this out then we don't have the time to implement it. Basically it's porting over some of the multimedia stuff that really belongs in the system.

6.3.15. Great 24-bit support

By this is meant that applications can just treat the system as if it were 24 bits and the right thing will happen on 16 and 8 bit systems. For example dithering to a standard 256 color palette, as in Gideons bible. Doing monochrome conversions in an intelligent fashion when going to printers.

6.4. NICE to Do

6.4.1. Cleanup

6.4.2. lines

6.4.2.1. dithered

6.4.2.2. cosmetic

6.4.3. brushes

6.4.3.1. patterns that wordk at different resolutions

6.4.4. text

6.4.4.1. dithered

6.4.5. widelines

6.4.5.1. look better

6.4.5.2. styles

6.4.5.3. dithered

6.4.5.4. caps\joins

6.4.5.5. square, round, etc.

6.4.6. line layout

We should just get code from another group and put it in Windows.

6.4.7. Transforms

NT has this, but I really haven't heard may requests for this from ISV's

7. USER

What do we need to enhance in USER for Chicago? After looking at the hundreds of feature requests and ideas I have received and then reviewing the stated goals for Chicago I have come up with three major catagories of work in USER. The following document serves as a high level outline for those three major areas which are expanded system capacity, a UI which is easier to use, and improved robustness and performance.

Looking a little closer into each of these three areas, we can outline five major areas of focus that will be of greatest benefit to the project. We are expanding system capacity by both eliminating the 64K heap limit as a barrier to applications and by removing all the other artificial restrictions that USER places on system resources. To make the UI easier to use we will add a new 3D look which provides consistency and better visual hints to the user and provide general support for the new shell. This also involves making aspects of the system much more configurable to the end user's needs and hardware capabilities. Finally, to improve robustness and performance we are implementing a new asynchronous input model, adding support for preemptive multitasking, and completing the validation layer so that all entry points into USER are protected.

7.1. SYSTEM CAPACITY

7.1.1. Support for expanded window and menu handle limits

Using a new 32 bit heap manager, window & menu handles will be mapped from their current 16 bit values, which were an offset into USER's data segment in Win 3.1, to a 32 bit far pointer internally. Hence, the 16 bit window or menu handle will be an index into its respective handle table, which will store the real 32 bit far pointer to the internal structure. This will provide for a 16K limit for both menu and window handles, which is far greater than the several hundred item limit present under 3.1.
(Oct. - Except for propogation of memory manager to kernel.)

7.1.2. Limitation on the number of Timers

Under Win3.1, the maximum number of Timers provided for all the apps and the system was 32 in total. This limit was imposed by a static array present in USER. With the increased demand for timer resources (the limit went from 16 to 32 timers between Win 3.0 and Win 3.1), we need to eliminate this limit entirely to provide a number of timers limited only by system memory resource limits.
(Done)

7.1.3. Limitation on the number of Ports supported

Under Win3.1, the maximum number of serial (COMM) ports supported was 9 and the maximum number of Parallel (LPT) ports supported was 4. For Chicago, we improve this limitation of the Communication API's to be consistent with the limit imposed by DOS. Hence, we will support up to 128 COMM ports and 128 LPT ports.
(Done)

7.1.4. Increased limit for listboxes

Previously, listboxes had a limitation of one 64k heap for the listbox item strings and one 32k heap for the listbox item data. This meant that a normal listbox with an average item string length of 10 characters, you could only have 6,400 items. And in the owner-draw case, you were still limited by the item data heap -- 8,192 items was the limit. Although this limit seems reasonable at first glance, there are many situations where we were informed by several ISV's that this limit was being reached during Win 3.1 development. For Chicago, the string heap and the item data heap can each grow into multiple segments. This eliminates these earlier item limits. The new limit is 32K items -- restricted by the fact that the index is passed in the wParam of a message and is therefore limited to a WORD, and we do a lot of operations relying on the index being signed, so that takes the limit from 64K to 32K.
(Done)

7.1.5. NT compatibility

In order to sustain a reasonable subset of the Win32 API it will be important to identify new API's that NT's USER supports and consider the benefit of porting these to the Chicago platform. One example of a set of API's that we know that we should port include some new API's to manipulate accelerator tables. These are (Create/Copy/Destroy)AcceleratorTable. There quite probably will be more API's as well.
(Post Oct.)

7.2. USER INTERFACE / EASE OF USE

7.2.1. Enhancement of the UI to provide a consistent 3D look

We will implement new 3D window border styles, control appearance, and color usage to provide a more esthetically pleasing and more psychologically consistent UI. This should make the system look better, be more intuitive and hence easier to u This will be achieved in part by consistent 3D appearance, the use of color to indicate static/editable fields and controls, and a more consistent look for disabled, depressed, hi lighted, selection, etc. appearance between controls.
(Oct.)

7.2.2. Scalable Window Metrics

There are many aspects of the components of the standard frame window which are based on simple standards and the system font size. However, there is no reason why many of the parameters which define the appearance of these window components could not be defined by the user. Just as we allow the setting of certain system colors and parameters for Win 3.1, we should expand this paradigm to allow the user to specify the size of the caption, menu, frame buttons, and scroll bars along with specifying the font for the caption and menu bar. This will allow each user to customize the appearance of their window to something that they are most comfortable with, and would be of special significance to those who are physically impaired and for use on portable machines that do not fit the standard desktop with a mouse setup.
(Oct. - except fine tuning to support the yet designed control panel interface)

7.2.3. DrawFrameControl / DrawBorder / DrawEdge

These are new API's for Chicago. DrawFrameControl provides applications with a way to draw part/all of a frame window via the common frame painting functions. This provides a way to have a consistent look in a Windows application that wants to be responsible for drawing it's own window frames. In addition, this API can also draw any of the system bitmaps at any size. We envision that this API could be expanded to draw almost any object that USER renders otherwise. This API is also the workhorse for support of the Scalable Window Metrics functionality above. DrawBorder and DrawEdge are new API's from Cairo which handle drawing the 3D shaded effects of controls and window frames.
(Oct.)

7.2.4. New color scheme

On 256 color devices, which are becoming increasingly popular in accelerated forms, we will implement a new color scheme such that the user will be able to select a base color for controls from which all the highlight and shadow colors will be generated. This will allow for a vast array of different yet very pleasing color schemes, besides the shades of gray we have implemented now. In prototype form this has been described as quite stunning visually.
(Post Oct.)

7.2.5. Proportional scroll bar thumbs

Windows scroll bars visually convey 2 pieces of information to the user: position and range. The user can use the scroll bar thumb to determine what percentage of the way they are through a file, for example. But one piece of information that we don't provide a visual clue for is the page size, i.e. the percentage of the range that is visible in the window at a given time. Scroll bars that provide this visual clue are said to have proportional scroll bar thumbs. Proportional scroll bar thumbs will be implemented in Chicago. Now, if an edit control has 100 lines of text, and the edit control window is large enough to show 10 lines of text, the scroll bar thumb takes up 10% of the scroll bar.
(Done)

7.2.6. Support of True Type fonts in Edit controls

Some of the True Type fonts have characters that have negative A and C widths. In other words, characters can overlap each other. The edit control code in USER was not aware of the possibility of negative A and C widths. So, when such fonts are used in edit controls under Win3.1, they looked very ugly because of undesirable clippings and erasings. For Chicago, this problem is fixed by rewriting some parts of edit control code to make it aware of the possible negative A and C widths in True Type fonts. See the reference document editctl.doc for more details.
(Done)

7.2.7. Dynamic display changes

With the proliferation of new display cards and video hardware, it is becoming increasingly important to allow the user to quickly change displays. Items that would be useful to implement would be the ability to dynamically change display resolution and/or depth. In addition we need to provide a standardized mechanism through which tilt screen displays can be supported.
(Post Oct.)

7.2.8. Ability to Get/Set Keyboard Layout

In order to facilitate the international user to enter one paragraph in French and another paragraph in German etc., we provide ways for programs to find out the current keyboard layout and change it dynamically. The SystemParametersInfo() API is expanded to do this.
(USER support done - need applet to actually support the different configurations and provide visual feedback.)

7.2.9. Shell support items

There are many aspects of the new Cairo shell design which have implications for support in USER. These items are tough to determine at this point, but may include:

- API's for better window/icon arranging
- New minimized look
- Small icons instead of system menus and used in other areas
- Virtual desktop support, possibly with different display resolutions and depth
- Bottommost window support
- Color and animated cursors
- Mini-caption windows
- Many miscellaneous UI and behavior changes

(Post Oct. - Except what is necessary to get Cairo running on Chicago)

7.2.10. Cougar support items

Cougar will require support in USER for fixing Alt-Tab from DOS boxes, save/restore functionality, and other miscellaneous minor work items.
(All current issues except save/restore done by Oct.)

7.3. ROBUSTNESS / PERFORMANCE

7.3.1. Asynchronous input model & serialization

One of the major complaints of Win 3.1 is the lack of "responsiveness" and "preemptive multitasking". What users really want from a windowing system is the ability to perform multiple tasks simultaneously efficiently. They need to be able to switch away from any application at will, and they do not want the poor performance or hanging of a single application to effect the system as a whole. These problems are in part solved by implementing a new asynchronous input model and serializing access to USER so that we can protect access to global data. With this scheme implemented along with preemptive multitasking, a 32 bit application will not be able to hang the system and you will be able to switch between applications whenever you want to. On the application development side, threads will be able to make calls to USER API's and the programming model will be identical to that of NT's.
(Post Oct.)

7.3.2. Exception handling

There are many places in USER where untimely GP faults or application termination can cause local memory allocations not to be freed. We need to be able to handle these exceptions properly and free memory in these situations.
(Post Oct.)

7.3.3. Complete validation of existing API's and exported but not documented API's

Many API's and Messages were not Validated by the validation layer under Win3.1. Now, for Chicago, we will validate such API's and Messages. Some API's are exported but not documented, and will likewise be validated. Also, we will improve the validation done for Cursor and Icon handles.
(Done)

HIGHLY CONFIDENTIAL

7.3.4. Dialog display time

This is still a big area in USER where we need to and can improve upon dramatically. This can be accomplished both by painting optimizations and possibly by the implementation of simpler window styles for controls so that there is much less unnecessary message traffic.
(Post Oct.)

MX 2120504
CONFIDENTIAL

CMS 00013798

RBC 04199

7.3.5. Protection of USER memory when not in USER semaphore

This would be a powerful debugging tool, using the memory manager to actually protect USER memory (our DS and Window/Menu heaps) so that if applications write on this memory we fault or display some debug information.
(Post Oct.)

7.3.6. Ownership of Menus

Under Win3.1, a menu created by the system was owned by the current task that created it. But, if that menu was not destroyed by that task before it terminated, it was left in the system memory for ever. Our attempts to clean up such things during task-termination time revealed some problems if the menu was created by a DLL and shared by multiple tasks.

Under Chicago, a menu is owned by the module that created it (rather than the task). If it is not destroyed by the module properly, then when the module is unloaded, the system will cleanup such menu structures from memory.
(Done)

7.3.7. New control features

7.3.7.1. LB_INITSTORAGE

A common complaint from ISV's is that adding items to a listbox is slow. This was obvious in applications that added thousands of items to a listbox. This message was added for listboxes to speed up bulk adds to a listbox. The wParam indicates how many items you're about to add. The lParam indicates the total byte count of the strings of all the items you're about to add. The test case of adding 32,000 items went from taking 48 seconds without this message to 13 seconds with this message.
NOTE There needs to be a CB_INITSTORAGE equivalent.
(Oct.)

7.3.7.2. LBS_NOITEMDATA

Listboxes have always kept a 4 byte data value for each item in the listbox. This is an expensive storage hit, and not all applications use these data values. So, for Chicago, we provide an LBS_NOITEMDATA listbox style that indicates that the application doesn't want the listbox to keep a 4 byte data value for each item.
NOTE There needs to be a CBS_NOITEMDATA equivalent.
(Oct.)

There will be more enhancements to other controls as well.

7.3.8. LoadStringHandle API

Currently, localization is difficult because of the size of static buffers in system components which store strings localized for different countries. It makes much more sense to simply provide an API which allocates the memory buffer for you and returns the requested string in this buffer.
(Oct.)

7.3.9. Remove 286 specific code

There is 286 specific code that can be removed from USER since Chicago requires a 386 or better to run on. This will decrease USER's size.
(Done)

7.3.10. Additional performance tuning

Besides some of the major work items outlined above, performance tuning will need to be done on an analysis bases. This is where we will need to run scenarios and benchmarks to find code fragments or API's that would most benefit from performance tuning.
(Post Oct.)

7.4. New API's, messages, styles, and parameters

This section contains duplication of the above material, but serves as a more detailed and central location providing all instances of changes in exported functionality from USER. THIS LIST IS BY NO MEANS COMPLETE OR TOTALLY ACCURATE! It serves as the base for the eventual SDK documentation.

7.4.1. int DrawFrameControl(HDC hdc, LPRECT lprc, DWORD dwType, DWORD dwState)

* hdc is the HDC into which the drawing is performed.
* lprc indicates the bouding rectangle for the drawing.
* dwType indicates the type of frame control to draw.

current values for dwType are:
DFC_MENU -- the frame bitmaps associated with menus
DFC_WINDOWSIZE -- the frame bitmaps associated with min/max/restore
DFC_SCROLLARROW -- the frame bitmaps associated with scroll bar arrows
DFC_BUTTON -- the frame bitmaps associated with check boxes & radio buttons.

* dwState indicates the state of the specified frame control type to draw.

current values for dwState are:
DFCS_MENUMAIN -- the main system menu frame bitmap.
DFCS_MENUCHILD -- the MDI child system menu frame bitmap.
DFCS_MENUARROW -- the hierarchical submenu arrow frame bitmap.
DFCS_MENUCHECK -- the menu checkmark frame bitmap

DFCS_SIZEMIN -- the minimize frame bitmap.
DFCS_SIZEMAX -- the maximize frame bitmap.
DFCS_SIZERESTORE -- the restore frame bitmap.

DFCS_SCROLLMIN -- an arrow that points in the minimum direction (up or left)
DFCS_SCROLLVERT -- a vertical arrow
DFCS_SCROLLMAX -- an arrow that points in the maximum direction (down or right)
DFCS_SCROLLHORZ -- a horizontal arrow
DFCS_SCROLLLINE -- the line beneath the scroll bar arrow (as seen in the combobox drop down arrow)
DFCS_SCROLLCOMBO -- the drop down arrow associated with a combobox

DFCS_BTNCHECK -- the checkbox frame bitmap
DFCS_BTNRADIO -- the radio button frame bitmap
DFCS_BTNCHECKED -- in the checked state
DFCS_BTN3STATE -- in the indeterminate state of a 3 state button
DFCS_BTNMONO -- the old style of having of packing all the radio buttons and check boxes in one monochrome bitmap
DFCS_BTNBOXES -- the old style of packing all of the radio buttons and check boxes in one bitmap

DFCS_INACTIVE -- inactive state
DFCS_PUSHED -- pushed down (pressed button) state

7.4.2. LB_INITSTORAGE / CB_INITSTORAGE

Functionality to make adding lots of items to a listbox or combobox much quicker.

wParam - The number of items you are adding.
lParam - The total byte count of all the items you will be adding.

7.4.3. LBS_NOITEMDATA / CBS_NOITEMDATA

New listbox and combobox style indicating that each item does not need the 4 byte data value for each item in the control.

7.4.4. void SetScrollPage(HWND hwnd, int fnBar, int iPage, BOOL fRepaint); int GetScrollPage(HWND hwnd, int fnBar, BOOL fRepaint); returns iPage

These functions provide the UI for setting/getting the page size for proportional scroll bars. This is done automatically for internal controls (comboboxes, MLE's, listboxes.) The functions are modeled after the Set/GetScrollPos API's.

fnBar - SB_CTL, SB_HORZ, SB_VERT

iPage - The number of scroll positions per page.

7.4.5. TabbedTextOut - New functionality

The lpnTabPositions can be specified as negative values, which provides for right justified text on the tab-stop as opposed to left justified normally.

7.4.6. EM_GETMARGINS / EM_SETMARGINS

EM_SETMARGINS
    wParam - Specifies which margins to set. It can be one or both of the following new flags:
        EC_LEFTMARGIN (0x0001), EC_RIGHTMARGIN(0x0002).
    lParam - If EM_LEFTMARGIN flag of wParam is set, LOWORD(lParam) specifies the
        leftMargin value to be set (in pixels).
        If EM_RIGHTMARGIN flag of wParam is set, HIWORD(lParam) specifies the
        right margin value to be set (in pixels).
EM_GETMARGINS
    wParam & lParam - Unused.
    returns left margin in LOWORD and right margin in HIWORD .

7.4.7. Accelerator Table manipulation functions from NT

```
CopyAcceleratorTable(HACCEL hAccel, LPACCEL lpAccel, int nCount);
CreateAcceleratorTable(LPACCEL lpAccel, int nCount);
DestroyAcceleratorTable(HACCEL hAccel);
```

7.4.8. New window and dialog styles for borders - from Cairo

```
// These are window edges
#define WS_EX_RAISEDOUTER    0x00000100L // (BDR_RAISEDOUTER << 2)
#define WS_EX_SUNKENOUTER    0x00000200L // (BDR_SUNKENOUTER << 2)
#define WS_EX_OUTER          0x00000300L

#define WS_EX_RAISEDINNER    0x00000400L // (BDR_RAISEDINNER << 2)
#define WS_EX_SUNKENINNER    0x00000800L // (BDR_SUNKENINNER << 2)
#define WS_EX_INNER          0x00000C00L

#define WS_EX_RAISEDEDGE     0x00000500L // (WS_EX_RAISEDOUTER | WS_EX_RAISEDINNER)
#define WS_EX_SUNKENEDGE     0x00000a00L // (WS_EX_SUNKENOUTER | WS_EX_SUNKENINNER)
#define WS_EX_ETCHEDEDGE     0x00000600L // (WS_EX_SUNKENOUTER | WS_EX_RAISEDINNER)
#define WS_EX_BUMPEDGE       0x00000900L // (WS_EX_RAISEDOUTER | WS_EX_SUNKENINNER)
#define WS_EX_EDGEMASK       0x00000F00L

// These are client edges
#define WS_EX_RAISEDOUTERC   0x00010000L // (BDR_RAISEDOUTER << 4)
#define WS_EX_SUNKENOUTERC   0x00020000L // (BDR_SUNKENOUTER << 4)
#define WS_EX_OUTERC         0x00030000L

#define WS_EX_RAISEDINNERC   0x00040000L // (BDR_RAISEDINNER << 4)
#define WS_EX_SUNKENINNERC   0x00080000L // (BDR_SUNKENINNER << 4)
#define WS_EX_INNERC         0x000C0000L

#define WS_EX_RAISEDEDGEC    0x00050000L // (WS_EX_RAISEDOUTERC | WS_EX_RAISEDINNERC)
#define WS_EX_SUNKENEDGEC    0x000a0000L // (WS_EX_SUNKENOUTERC | WS_EX_SUKNENINNERC)
#define WS_EX_ETCHEDEDGEC    0x00060000L // (WS_EX_SUNKENOUTERC | WS_EX_RAISEDINNERC)
#define WS_EX_BUMPEDGEC      0x00090000L // (WS_EX_RAISEDOUTERC | WS_EX_SUNKENINNERC)
#define WS_EX_EDGEMASKC      0x000F0000L
```

```
// Default top-level window extended style
#define WS_EX_OVERLAPPEDWINDOW    (WS_EX_RAISEDEDGE | WS_EX_SUNKENEDGEC)

// Dialog styles and edges

#define DS_RAISEDOUTER          0x1000L
#define DS_SUNKENOUTER          0x2000L
#define DS_OUTER                (DS_RAISEDOUTER | DS_SUNKENOUTER)
#define DS_RAISEDINNER          0x4000L
#define DS_SUNKENINNER          0x8000L
#define DS_INNER                (DS_RAISEDINNER | DS_SUNKENINNER)

#define DS_RAISEDEDGE           (DS_RAISEDOUTER | DS_RAISEDINNER)
#define DS_SUNKENEDGE           (DS_SUNKENOUTER | DS_SUNKENINNER)
#define DS_ETCHEDEDGE           (DS_SUNKENOUTER | DS_RAISEDINNER)
#define DS_BUMPEDGE             (DS_RAISEDOUTER | DS_SUNKENINNER)
#define DS_EDGEMASK             0xF000L
```

7.4.9. BOOL DrawBorder(HDC hdc, LPRECT grc, BDR bdrType, UINT grfFlags); - From Cairo

```
// 3D border styles
typedef UINT BDR;

#define BDR_RAISEDOUTER 0x0001L
#define BDR_SUNKENOUTER 0x0002L
#define BDR_RAISEDINNER 0x0004L
#define BDR_SUNKENINNER 0x0008L

#define BDR_OUTER     0x0003L
#define BDR_INNER     0x000cL
#define BDR_RAISED    0x0005L
#define BDR_SUNKEN    0x000aL

// Border flags
#define BF_LEFT       0x0001L
#define BF_TOP        0x0002L
#define BF_RIGHT      0x0004L
#define BF_BOTTOM     0x0008L

#define BF_TOPLEFT     (BF_TOP | BF_LEFT)
#define BF_TOPRIGHT    (BF_TOP | BF_RIGHT)
#define BF_BOTTOMLEFT  (BF_BOTTOM | BF_LEFT)
#define BF_BOTTOMRIGHT (BF_BOTTOM | BF_RIGHT)
#define BF_RECT        (BF_LEFT | BF_TOP | BF_RIGHT | BF_BOTTOM)

#define BF_SOFT                 0x1000L       // For soft buttons instead of tiles
#define BF_FLAT       0x2000L  // For push buttons that press flat
#define BF_MIDDLE     0x4000L  // Fill in the middle
#define BF_ADJUST     0x8000L  // Calculate the space left over
```

7.4.10. BOOL DrawEdge(HDC hdc, LPRECT grc, EDGE edgeType, UINT grfFlags);

```
// 3D Edge style
typedef UINT EDGE;
```

```
#define EDGE_RAISED    (BDR_RAISEDOUTER | BDR_RAISEDINNER)
#define EDGE_SUNKEN    (BDR_SUNKENOUTER | BDR_SUNKENINNER)
#define EDGE_ETCHED    (BDR_SUNKENOUTER | BDR_RAISEDINNER)
#define EDGE_BUMP      (BDR_RAISEDOUTER | BDR_SUNKENINNER)
```

7.4.11. HWND WindowFromDC(hdc); - From Win32

WindowFromDC returns the window associated with the passed display context. Any output functions to this display context will draw into this window.

Parameter	Description
hdc	Identifies the display context whose associated window is to be retrieved.

Returns

If the function succeeds, the return value identifies the window associated with the given display context. Otherwise, the return value is NULL.

See Also

GetDC

8. Win32/Kernel32

8.1. Goals

The Win32/Chicago subsystem has two key goals:

- Threads/Preemption/Desynchronized input support. This is our number one priority and feature and is all the same ball of work. This will allow our users the 'smooth multitasking' they have been asking for and will allow them to switch away from 'hung' or busy applications. This will give us the great background printing we've been after.

- Run NT applications. We are seeking to provide a consistent platform for ISVs to make powerful programs for NT and have them run on a commodity platform that sells millions. We want to give ISVs and end-users the best of both worlds here. I realize that this is a large set of features, but this is a criteria that lets us define which APIs to implement beyond Win32s: does the API help us run garden-variety NT programs and provide a consistent programming platform?

8.2. Introduction

Chicago provides a subset of the Win32 API that is binary compatible with the same subset on the Windows NT x86 platform. The Chicago Win32 API is also a superset of the Win32s API, and is upwardly binary compatible with Win32s. This document describes the architecture and limitations of Chicago's Win32 support, called Win32/Chicago.
The basic model for supporting the Win32 API is the same model used by Win32s: 32-bit API's are thunked wherever possible to their corresponding 16-bit Windows 3.1 API's. For Win32 API's that do not have 16-bit counterparts or where the 16-bit semantics impose too great of a restriction, new 32-bit code is used and the API's are not thunked. For example, most of the USER32 and GDI32 API's are implemented via thunks, but most of the KERNEL32 API's are implemented in pure 32-bit code.

Unlike Win32s which must run on the installed base of Windows 3.1 and therefore cannot make any changes to the 16-bit components on which it is dependent, Win32/Chicago is free to modify and enhance the 16-bit Windows components wherever appropriate. An example of this is the modification to USER that desynchronizes the input model in order to provide a smoother multitasking user interface and greater compatibility with the Win32 API.

8.3. Terminology

Console The category of Win32 API's that allow non-GUI 32-bit applications to read keyboard input and display text in a "console" window. Example console applications are compilers, linkers, etc.

Cougar MSDOS with protected mode support. Provides low level protected mode support for Win32.

DLL Dynamic Link Library. DLL's share the same module format as EXE's, with NE EXE's linking to NE DLL's, and PE EXE's linking to PE DLL's (MZ DLL's do not exist). Note that NE and PE modules cannot be interlinked. Some of the Chicago DLL's referenced in this document:

KERNEL	16-bit
GDI	16-bit
USER	16-bit
KERNEL32	32-bit
USER32	32-bit (thunk layer)
GDI32	32-bit (thunk layer)

EXE Executable program module. Chicago supports three different exe module formats: MSDOS (MZ), Windows (NE), and Win32 (PE).

Thunk A piece of code that translates parameters passed to a 32-bit function into a form suitable for a corresponding 16-bit function (or vice versa).

Win32 32-bit Windows API. See the Win32 API Guide for details.

8.4. Win32 Components

HIGHLY CONFIDENTIAL

Services for supporting the Win32 API come from several different components of Chicago:

- Cougar

MX 2120510
CONFIDENTIAL

CMS 00013804

RBC 04205

- Virtual memory management
- Dos protected mode interface (DPMI)
- Installable file systems (IFS)
- Thread management
- Win32 VxD
- Structured exception handling (SEH)
- Debugging support
- Thread services
- 32-bit API services
- Portable Exe loader
- Dynamic linking
- Process/thread management
- Memory management
- Floating point emulation
- Structured exception handling
- Thunk management
- 32-bit API thunk libraries
- Thunks
- Windows 16-bit API dynamic link libraries
- USER API
- GDI API
- DDEML
- COMMDLG
- et. al.

8.4.1. Low Level Support Services

Win32 makes use of low level services provided by Cougar. Although Cougar can run without Win32, Win32 is very dependent on Cougar and will not run on any other version of MSDOS. Virtual memory management, file system support and protected mode thread management (creation, termination, scheduling) are the primary services provided by Cougar to support the Win32 API.

Other low level support comes from the Win32 VxD supplied with Chicago. This VxD provides protected mode support for structured exception handling and debug API's, as well as the initial "bootstrapping" process of loading the 32-bit system dll's.

8.4.2. Kernel

The Win32 kernel provides the core 32-bit functionality for the KERNEL32 API's as well as support for the 32-bit API thunk libraries.

8.4.2.1. Startup and Initialization

The Win32 VxD is dynamically loadable and unloadable with its loading being controlled by the 16-bit WinExec and LoadModule API's. If a Windows application requests to run a Win32 application, KERNEL will recognize this and load the Win32 VxD if it's not already loaded. Loading the VxD begins the startup and initialization phase.

Once the Win32 VxD is loaded, it automatically loads KERNEL32 and the thunk DLL's. These 32-bit system DLL's are simple memory mapped modules, and the entire process of loading them involves nothing more than mapping them into memory. No fixups are required since the Cougar memory manager guarantees the load addresses of the system DLL's, and all imports are resolved statically at build time.

When the system DLL's have been successfully loaded, control returns to KERNEL where it then calls the initialization entry point in KERNEL32 via a 16->32 thunk. When initialization has completed, KERNEL is then able to use whatever KERNEL32 functions it chooses.

8.4.2.2. Portable Exe File Loader

Win32 applications use the Portable Exe (PE) format. The PE loader in KERNEL32 handles loading EXE and DLL files by allocating the virtual memory for the modules, resolving import references and relocation fixups (if necessary), and supporting demand paging.

All PE modules have a preferred load address that is set at link time, and when loaded at this address, relocation fixups do not need to be performed. On the other hand, if they cannot be loaded at the preferred address (because of a conflict with another DLL or memory allocation), then the module is loaded at any available address and fixups are applied to individual pages when they are demand loaded[8]. As stated above, the 32-bit system DLL's are guaranteed to be loaded at their preferred load address, and since Win32 applications execute in separate address spaces, EXE modules are also guaranteed to load at their preferred address (because this address is the first allocation in their private memory arena).

8.4.2.3. Process Management

A process has its own memory context and is an owner of resources such as file handles and memory blocks. When a process terminates, all of the resources used by the process are freed, making them available to other processes in the system. A process will always have at least one thread associated with it. A process generally represents a single application program.

Each Win32 application executes in its own address space that is separate from all other applications in the system, including 16-bit Windows applications. The address space of Win32 applications is divided into two arenas: the private arena and the shared arena. All memory that is dynamically allocated by the application resides in the private arena[9], as does the EXE module itself. The shared arena is global to all processes, both 32-bit and 16-bit, and is the arena where all 32-bit DLL's[10] are loaded as well as all 16-bit EXE's and 16-bit DLL's.

A process is created by creating a new memory context and thread (via Cougar) and initializing the appropriate user mode data structures in KERNEL32. The thread begins executing in the new process context on a temporary stack allocated in the shared arena, and one of the first actions taken by the thread is to load the specified EXE module. Once loaded, the thread switches to its permanent stack, appropriate library initialization routines are called, and then control is transferred to the application's entry point as specified in the EXE.

A process is terminated in one of two ways. If ExitProcess is called, then any loaded DLL's attached to the process are notified by calling their library termination routines if required. Otherwise, in the case of TerminateProcess, library termination routines are not called, and the process is terminated without ever calling into application code (EXE or DLL) again. In both cases, the Win32 system DLL's are notified upon process termination to allow for cleanup, e.g. to free any resources allocated on behalf of the process.

8.4.2.4. Thread Management, Scheduling and Synchronization

A thread represents a single thread of execution and is the basic entity for scheduling. Threads are owned by a single process and do not change ownership. Each thread has a priority and state associated with it that is used by the scheduler. Low level support for thread creation and termination is provided by Cougar.

The scheduler performs preemptive, priority based, round-robin thread scheduling. Preemption is based on a fixed time slice. When the time slice has expired, the current thread's state is saved, and the next runnable thread's state is restored. There are five priority levels, -2 to +2, and any threads that are runnable at a given priority exclude lower priority threads from running. Threads at the same priority are run in a round-robin fashion, i.e., the thread that has been waiting the longest to be run will be run next.

Synchronization objects are used to synchronize operations between multiple threads, either within the same process, or across processes. There are two primary uses for synchronization objects: 1) to protect a resource from being used by more than one thread at a time, and 2) for a thread to signal another thread that a particular event has occurred. Win32 defines

[8]This differs from the method used in Windows NT for applying fixups. See Appendix B for details.
[9]Memory allocated via GlobalAlloc with the GMEM_DDESHARE flag is allocated in the shared arena. See Appendix B for details.
[10]This is an area that differs in implementation between Windows NT, Win32s and Chicago. See Appendix B for details.

four synchronization objects: Critical Sections, Mutexes, Events, and Semaphores. Critical Sections, Mutexes, and Semaphores are used to protect resources, and Events are used to signal that an event has occurred.

8.4.2.5. Memory Management

Chicago contains several different memory managers, with the Cougar virtual memory manager providing the foundation for all of the others. KERNEL contains the 16-bit Windows Global and Local memory managers that are responsible for the 16-bit GlobalAlloc and LocalAlloc series of API's, respectively. The Win32 series of GlobalAlloc, LocalAlloc, VirtualAlloc, and HeapAlloc API's are contained in KERNEL32 and are implemented fully in 32-bit code, i.e. they are not thunked to the 16-bit API's.

The HeapAlloc, LocalAlloc and GlobalAlloc API's are all implemented via a single 32-bit heap manager. The heap manager uses a multiple free list algorithm where free blocks are partitioned into many doubly linked lists and each list is sorted by size, largest block first. An allocation request first considers the initial element of the list that contains elements that may or may not fit. If this element does not fit, then the last element of the next higher up non-empty list is used. The allocation request is satisfied by splitting this element into a used block and a free block which is then placed on the appropriate free list. Free blocks are coalesced and sorted at free request time.

8.4.2.6. Thunk Selector Management

Thunking 32-bit functions to 16-bit functions requires that 0:32 flat addresses be converted to 16:16 segmented addresses. Since Win32 does not automatically tile any memory allocations with selectors, the mapping of flat addresses to segmented addresses requires the allocation and initialization of a descriptor to create the 16:16 mapping[11].

The thunk manager must deal with the following restrictions and requirements:

1. Selectors are a limited resource.
2. The lifetime of a 16:16 thunk mapping is not always known, i.e. some mappings need to exist for the duration of the underlying memory object.
3. Memory regions larger than 64K must be thunked by mapping consecutive (tiled) 64K selectors.
4. The general thunk mechanism must be as fast as possible since it is used on virtually every category of API.

The limited number of selectors available will not be a problem for the thunk manager if all thunks are serialized for the duration of the thunked call. On the other hand, if some commonly called 16-bit API's are made reentrant for both 16-bit and 32-bit threads, then the number of selectors that can be consumed at any particular time is unbounded.

The great majority of thunks that create linear to segmented mappings on entry destroy those mappings on exit. However, a few API's require that the mappings remain until some time after the call returns. For these persistent mappings, the thunk manager cannot destroy the mapping until the underlying memory object is itself freed. All persistent mappings are tracked by a mechanism that utilizes a hash table to prevent multiple persistent mappings to the same region of memory. On every allocation of a persistent mapping, the hash table is searched for an existing mapping, and if found, the existing mapping is reused. If a mapping does not already exist, then one is created and added to the hash table.

The creation of linear to segmented mappings is made as fast as possible by keeping it as simple as possible. The most common mapping is one that is less than 64K in size and is not persistent. For this case, the mapping is performed in one function by first allocating a selector (done simply by removing the initial element from a singly linked list of free descriptors), and then initializing the descriptor appropriately. This requires fewer than two dozen instructions when written is assembler. For persistent mappings or mappings requiring tiling, these cases are rare and therefore do not need to be hand optimized.

<u>8.4.3. Console</u>

TBD

MX 2120513
CONFIDENTIAL

<u>8.4.4. Debugging Support</u>

[11]This differs in implementation from Win32s. On that system, most memory dynamically allocated by the application is allocated with tiled selectors already mapping it. The thunk manager then has to search for the appropriate selector mapping the memory.

The goal is to provide the full Win32 debug API. Details TBD.

8.4.5. OLE

TBD

8.4.6. Network Support

TBD

8.4.7. Mailslots

TBD

8.4.8. Communications

TBD

8.4.9. Pipes

8.4.9.1. Anonymous

TBD

8.4.9.2. Named

TBD

8.4.10. Structured Exception Handling

An exception is an event that occurs during the execution of a program which requires the execution of software outside the normal flow of control. Exceptions can result from the execution of certain instruction sequences, in which case they are initiated by hardware. Other conditions may arise as the result of the execution of a software routine (e.g., an invalid parameter value), and are therefore initiated explicitly by software.

When an exception is initiated, a systematic search is performed in an attempt to find an exception handler that will dispose of (handle) the exception. An exception handler is a function written to explicitly deal with the possibility that an exception may occur in a certain sequence of code. Exception handlers are declared in a language-specific syntax and are associated with a specific scope of code. The scope may be a block, a set of nested blocks, or an entire procedure or function. Microsoft compilers for Win32 adhere to a common calling standard which enables exception handlers to be established and disestablished in a very efficient manner.

The goals of the Win32 exception handling capabilities are the following:

- Provide a single mechanism for exception handling that is usable across all languages.
- Provide a single mechanism for the handling of hardware-, as well as software-generated exceptions.
- Provide a single mechanism for the handling of exceptions and for the capabilities necessary to support sophisticated debuggers.
- Provide an exception handling mechanism that is portable and which separates machine-dependent from machine-independent information.
- Provide an exception handling mechanism that supports the structured exception handling extensions being proposed by Microsoft for the C language (see Structured Exception Handling in C by Don MacLaren, May 10, 1989).

8.4.10.1. Exception Architecture

The overall exception architecture of Win32 encompasses the process creation primitives, system service subsystems, the Microsoft calling standard(s), and system routines that raise, dispatch, and unwind exceptions. One optional exception port, the debugger port, may be specified when a process is created.

When an exception is initiated, an attempt is made to send a message to the recipient process's debugger port. If there is no debugger port, or the associated debugger does not handle the exception, then a search of the current thread's call frames is conducted in an attempt to locate an exception handler. If no frame-based handler can be found, or none of the frame-based handlers handle the exception, then another attempt is made to send a message to the recipient process's debugger port. If there is no debugger port, or the associated debugger does not handle the exception, then the system provides default handling based on the exception type.

Thus the search hierarchy is:

- Debugger first chance

- Frame-based handlers
- Debugger second chance

8.4.10.2. Frame-Based Exception Handlers

An exception handler can be associated with each call frame in the procedure call hierarchy of a program. This requires that each procedure or function that either saves nonvolatile registers or establishes an associated exception handler, have a call frame.

Microsoft compilers for Win32 adhere to a standard calling convention for the construction of a call frame. The exact details of the call frame layout are described in the Microsoft Win32 calling standard(s).

8.4.10.3. Exception Dispatching

When a hardware exception occurs, the Win32 trap handling software gets control and saves the hardware state of the current thread in a context record. The reason for the trap is determined, and an exception record is constructed which describes the exception and any pertinent parameters. Kernel software is then called to dispatch the exception.

The exception dispatcher attempts to send a message to the associated debugger port. This message includes the exception record and the identification of the client thread. The debugger may handle the exception (e.g., breakpoint or single step) and modify the thread state as appropriate, or not handle the exception and defer to any frame-based exception handlers found on the user stack.

If the debugger replies that it has handled the exception, then the machine state is restored and thread execution is continued. Otherwise, if the debugger replies that it has not handled the exception, or there is no debugger port, then kernel software must prepare to execute the exception dispatcher in user mode.

If the debugger does not dispose of the exception, then stack space is allocated on the user stack, and both the exception record and the context record are moved to the user stack. The machine state of the thread is modified such that thread execution will resume in code that is part of the kernel, but it executes in user mode.

The machine state is restored and execution of the thread is resumed in user mode within kernel code that calls the exception dispatcher to search the user stack for an exception handler. If a frame-based handler handles the exception, then thread execution is continued. Otherwise, if no frame-based handler is found, or no frame-based handler handles the exception, then the LastChance system service is executed.

The purpose of the LastChance system service is to provide the debugger a second chance to handle the exception. A second attempt is made to send a message to the associated debugger port. This message includes the exception record and the identification of the client thread. The debugger may handle the exception (e.g., query the user and receive a disposition) and modify the thread state as appropriate, or not handle the exception and defer to any default handling supplied by the kernel, which in most cases causes the thread to be terminated.

8.4.10.4. Exception Handling and Unwind

During the dispatching of an exception, each frame-based handler is called specifying the associated exception and context records as parameters. The exception handler can handle the exception and continue execution, not handle the exception and continue the search for an exception handler, or handle the exception and initiate an unwind operation.

Handling an exception may be as simple as noting an error and setting a flag that will be examined later, printing a warning or error message, or taking some other overt action. If execution can be continued, then it may be necessary to change the machine state by modifying the context record (e.g., advance the continuation instruction address).

If execution can be continued, then the exception handler returns to the exception dispatcher with a status code that specifies that execution should be continued. Continuing execution causes the exception dispatcher to stop its search for an exception handler. The machine state from the context record is restored and execution is continued accordingly.

If execution of the thread cannot be continued, then the exception handler usually initiates an unwind operation by calling a system-supplied function specifying a target call frame and a continuation address. The unwind function walks the stack backwards searching for the target call frame. As it walks the stack, the unwind function calls each exception handler that is encountered to allow it to perform any cleanup actions that may be necessary (e.g., release a semaphore, etc.). When the target call frame is reached, the machine state is restored and execution is continued at the specified address.

8.4.11. Asynchronous I/O

8.4.12. Printing

TBD

8.5. Enhancements to 16-bit Windows

8.6. Unsupported Features

8.7. Appendix A: Supported Win32 API's

8.8. Appendix B: Issues

9. Communications Support

This section discusses communications support in the Chicago Windows release.

9.1. Overview

In 1992, communications in Windows is similar to printing for MS-DOS applications. ISVs are responsible writing for most of tl complex code and user interface, and application implementations are of generally low quality, with non-standard user interfaces and significant performance limitations.

With Chicago, Windows must earn a reputation as the platform of choice for all forms of communications, "desktop" and otherwise, including e-mail, fax, on-line services, telephony, and more. This will require an effort similar to that applied to Windows 3 1 for its printing advances.

This document discusses requirements and design for communications hardware and application support in the Chicago Windows release. The 1.0 version is a high-level discussion of feature requirements. This document will grow and change as features are added, deleted, and designed.

9.1.1. The 5 Chicago Improvements

Chicago's primary improvement areas apply to communications features as follows:

- Plug & Play: Solve hardware conflict problems experienced by Windows 3.1 customers. Eliminate the need for users to configure address and IRQ settings in Control Panel, SYSTEM.INI, etc.
- New User Interface/Easier to Use: Allow users to make connections with minimal configuration of ports, modems, protocols, etc.
- Integrated and Complete Protect Mode System: Communications port and protocol drivers will run at ring 0, if necessary. Other protected mode device support will allow baud rates not possible in real mode MS-DOS.
- Great Network Client and Peer Server: Chicago should provide access to remote comma devices and the ability to share local comma devices on the network. Plus, remote access capabilities should be provided by modem. The Chicago Strategy Document states,

Comm sharing/redirection: The ability to share and redirect comm ports and modems in a peer-to-peer networking environment is an obvious thing to do.

Dialup support for networking: Chicago will utilize the phone as the network. Enabling dialup from anywhere there are phones and modems.

- 32-bit application support: Big open question here! Is the Windows NT communications API supported in Win32/S for Chicago?.

9.1.2. The 5 Chicago Requirements

Chicago's primary technical requirements apply to communications features as follows:

- Compatibility: Windows 3.x communications applications and drivers will run under Chicago.
- Great 4 Mb System: Drivers are dynamically loadable.
- Performance: Provide equivalent or better serial communications throughput for MS-DOS and Windows applications running on 386/20 based PCs on 16550A UARTs.
- Robustness: Provide reliable 19,200 baud serial communications support for MS-DOS and Windows communications applications running on 386/20 based PCs on 16550A UARTs.
- On Schedule: Of course.

9.1.3. Other Objectives

In addition to the 10 Chicago improvements and requirements, communications features should include the following:

- Improved API: Answer the requests of ISVs in order to obsolete the need for third party communications libraries. These are common under MS-DOS to support non-standard devices, and provide basic protocol and modem functions. Under Windows, add-on third party products for enhanced communications API will cause grief down the road.
- Broader Device Support: Remove limitations to standard COM and LPT ports in order to accommodate intelligent adapters, high-speed and bi-directional parallel ports, etc.
- Integrated Fax Support: The Chicago Strategy Document states,

FAX management: The system will incorporate FAX management as a basic system function, just like email and printing. This includes a simple FAX applet and close integration with email and the forms support built into Chicago.

9.1.4. Non-Objectives

Chicago will not attempt to achieve the following communications-related objectives:

- Windows NT Model: Chicago will not attempt to provide a superset of the Win/NT communications functionality. Specifically, the following Win/NT features will not be supported in Chicago:

Queued Allocation of Devices: An application will not be able to automatically wait for an busy communications resource. Instead, an open call will fail, and the application and/or user will have to retry later.

Destination Resource Enumeration: Communications device drivers will register destination resources with a resource catalog manager, e.g. LAT servers, etc.

9.1.5. Important Features

This specification discusses the following important features for Chicago:

- Windows 3.1 Compatibility
- Solve Port Address/IRQ Problems: Fix common installation problems.
- 16550A UART FIFO Support: Improved throughput for MS-DOS comm apps.
- Kernel Improvements: Make Chicago a "comm friendly" OS.
- Improved Device Contention: A better UI for using ports from multiple applications.
- Installable Port Drivers: Ability to install drivers for non-standard comm devices.
- Bi-Directional Parallel Port Support
- Modem Status Information API: GetCommEvent()
- Using Remote Communications Devices
- Sharing Local Communications Devices
- Protocol API & Drivers
- Fax Management

Additional features under consideration for development:

- Modem API & Drivers
- Communications Manager
- Communications Setup Dialog
- Auto-Negotiate Protocol

Major Open Issues

- Who owns development of Chicago communications features?
- Should usability testing be done?
- Should communications device drivers be incorporated into Dragon?
- Should the WfW team own local COM port sharing?
- Should the Utilities team own Fax Management?
- Cougar team should own Int14 support.

9.1.6. Related Documents

- DOS/Win32 (Cougar) Communication Driver Specification (COMM-API.DOC): David Thielen's proposal for a layered port, protocol, and modem device driver interface and API.
- Microsoft Windows 4.0 (Win/N) Communications Service Functional Specification: Lester Waters' communications API, DDI, and UI specification for Windows NT.

9.1.7. Revision History

09/25/92 Original 1.0 version submitted for review.

09/29/92 Added "Important Features" section.
Removed "On-Line Information Access", this goes to the Utilities team.

9.2. Competitive Analysis

Discussions of the communications capabilities of major competitive PC software platforms.

9.2.1. IBM OS/2

In September 1992, a scan of over 300 PC industry trade journals for discussions of OS/2 communications capabilities vs. Windows did not yield much indication that OS/2 is considered a stronger platform for communications than Windows. This is probably because most users do not require high-speed communications, and those that do are comparing Windows 3.1 with OS/2 on PCs which are fast enough to make the two relatively equal in most situations.

PC Magazine's 4/28/92 review, "OS/2 2.0: does it fulfill the promise?", mentioned communications capabilities indirectly in the following performance note:

Unfortunately, it is not as easy in Windows 3.1 to achieve such smooth and reliable multitasking during background communication. In fact, one particular incident in June, 1992 demonstrated that communications is indeed a likely Achilles heel for Windows in comparisons against OS/2.

The Louderback Incident

Shortly after the release of Windows 3.1, Jim Louderback wrote in PC Week that OS/2 made a better platform for a traveling PC because it was demonstrated to be more capable of reliable communications, even at the low speed of 2400 baud! After examining what Louderback was attempting to do, it became clear why OS/2 came out ahead. Louderback's procedures for "on the road communications" included starting and exiting several MS-DOS VMs for Commander, TapCIS, MCI Mail, and other applications while text downloads were performed in the background. He stated, "Since I am uploading and/or downloading almost continuously while working on columns or answering mail I cut my morning mail time in half..."

Errors similar to his were easily reproduced here at MS if a VM was exited during a background text download, even at 2400 baud. Shutting down a VM disabled interrupts long enough to cause character overruns at the port. Since text was being transferred, error correction was not applied, and message data was lost.

So, why did OS/2 work better than Windows 3 1 for Jim Louderback? On June 16, 1992 Aaron Reynolds answered this question as follows. .

In particular, there is a known problem in Windows 3 1's VM shutdown code:

What needs to change in Windows to make it better than OS/2?

Fortunately, that sounds a lot like Cougar. We can concentrate on Cougar to insure that it is well-behaved, i e. not disabling interrupts for too long. But how can we protect against third party code that interferes with background transfers?

So, Chicago should encourage usage of error-correcting protocols. We can do this by making it easier for ISVs to write applications which use them -- perhaps a simple protocol API.

9.2.2. Apple Macintosh

In 1989, Apple introduced the Macintosh Communications Toolbox, which was later incorporated into System 7.0 in 1991. The purpose of the Communications Toolbox was to make it easier for ISVs to write communications applications. It includes the following modular "manager" APIs:

Connection Manager	Basic connection services allowing applications to make connections without having to take into account underlying connection parameters. Modular connection tools handle selection of phone number, modem type, baud rate, flow control, error correction, error detection, data encapsulation, etc.
Terminal Manager	Generic terminal emulation services allowing applications to communicate with remote computers independent of the terminal emulator protocols required. Terminal emulator tools do the work for character and key code translations, etc.
File Transfer Manager	Allows application to send and receive files from remote computers independent of underlying file transfer protocols. File transfer tools are responsible for implementing protocols and configuring options.
Communications Resource Manager	Provides standardized routines and data structures to allow applications to keep track of communications devices users have installed. Used to select available communications resources, and resolve conflicts when two or more applications need to use the same resource at the same time.
Telephone Manager	Introduced November 1991. Standardized routines to access telephony features.
Modem Tool	Introduced March 1992. Standardized routines to access high-speed modem features.

The Mac Communications Toolbox has become a controversial topic among Mac ISVs. Although introduced in 1989, tools which plug into the various managers have been slow in coming from third party vendors. Also, ISVs have complained that Apple is not committed, and has taken too long to fix problems. There is speculation that the Communications Toolbox is being scrapped by Apple, which chose not to use it for its own AppleTalk Remote Access product.

These events signal that the Communications Toolbox was a somewhat misguided endeavor on the part of Apple. Chicago should meet the biggest needs that the Communications Toolbox was intended to provide for ISVs, but we should learn from Apple's lesson that ISVs are not flocking to build tools or incorporate communications functions into their applications. The reason for this is probably that the Communications Toolbox was built almost entirely from an ISV perspective. As a result, end-users are not isolated from things that make communications difficult in the first place, e.g. understanding baud rates, parity, protocols, etc. The Communications Toolbox simply shifts difficult user interface out of applications and into tools.

Specifically, Chicago should provide functionality that meets and exceeds the Macintosh in the following areas:

- Connection: Applications must be able to connect easily to any communications resource, local or remote, serial or parallel.

- File Transfers: Chicago should make it easy for users and applications to use binary transfer protocols, which speed data transfer and provide error correction.
- Modems: Applications must be able to use modems independent of command sets and extended capabilities.
- Telephones: A good telephone API is needed, but should sit on top of the Chicago communications API. A "TAPI" effort is already underway at Microsoft, this should be incorporated into Chicago.

9.3. Communications Usage Scenarios

This section examines customer usage of communications resources in Windows in an attempt to identify problem areas that need correction and opportunities for moving Windows' capabilities forward. Each of the following sections discusses common communications related activities that Windows customers experience and identifies problems they encounter. Later sections detail features that Chicago may include to address these problems.

The following problems are identified in this section, and the corresponding Chicago features which address them are listed:

Problem	Corresponding Features
Accessing Multiple Ports Simultaneously	Improved Device Contention
Port Addressing and IRQ Problems	Solve Port Address/IRQ Problems
Third Party Comm Driver Conflicts	Installable Port Drivers
Selecting the Correct Port	Improved Device Contention Plug & Play Communications Setup Dialog Using Remote Communications Devices Sharing Local Communications Devices
Configuring the Port	Plug & Play Installable Port Drivers Auto-Negotiate Protocol Communications Setup Dialog
Configuring the Modem	Modem API & Drivers Communications Setup Dialog
Sharing One Modem With Multiple Applications	Improved Device Contention
New Port Types Are Coming	Installable Port Drivers
Data Loss During Text Transfers	Kernel Improvements 16550A UART Support
Accessing Info Services, BBS's	On-line Information Access
Slow Throughput, Data Loss During File Transfers	Kernel Improvements 16550A UART Support Protocol API & Drivers
Optimizing File Transfers is Difficult	Auto-Negotiate Protocol
Determining Modem Status	Modem Status Information API

Device Selection & Contention	Communication Setup Dialog Improved Device Contention
API Too Device Dependent	Communication Setup Dialog Modem API and Drivers Protocol API and Drivers

9.3.1. Installing Communications Devices

Most PCs include COM1 and many now include COM2 in standard configurations. In addition to these, the most common serial devices installed by customers are as follows:

- Port Adapters: Many are pre-configured for COM3, since this is the most common available port. Customers frequently purchase these to connect secondary external modems or specialty communications devices.
- External Modems: Installation is easy, since these simply plug into an already installed port with a standard cable. These modems usually work immediately, provided that the port they are plugged into does not conflict with other serial devices.
- Internal Fax/Modem Adapters: These are becoming increasingly popular, but are usually problematic for Windows customers due to port address and IRQ conflicts. A common default is COM4, which causes problems when installed into PCs with no COM3.
- Multi-Port and Intelligent Adapters. Customers with special needs for more than four COM ports or high-speed transmission may install specialty adapters with non-standard hardware. These adapters are not supported by the standard Windows COMM.DRV.

The following problems are encountered by users when they install new communications devices:

- Accessing Multiple Devices Simultaneously: Users are confused when they cannot use two serial devices at the same time in Windows. For example, they may install a serial mouse on COM2 and fax modem on COM4. Both ports use IRQ3 on standard ISA hardware, and will conflict if used simultaneously. Since COM1 and COM3 usually share IRQ4, and COM2 and COM5 usually share IRQ3, customers which attempt to use these ports simultaneously encounter problems, and Windows does not warn of this conflict. On some EISA and MCA PCs, Windows can share IRQs, avoiding this conflict.

  Chicago should automatically use shared IRQ ports whenever possible, without requiring the user to put "COMIrqSharing=TRUE" in SYSTEM.INI. If a conflict is detected when a port is opened, a warning message should be generated allowing the user to continue or abort.

- Port Addressing and IRQ Problems: Windows has problems when COM devices are installed out of sequence. For example, it is common for fax/modem adapters to install as COM4 by default. If no COM2 or COM3 is present in the PC, Windows may assign an incorrect addresses and IRQ, or sometimes may not default at all. This causes a variety of symptoms, including:

  Serial mouse doesn't work in enhanced mode.
  Serial mouse doesn't work after a full-screen VM.
  Can't access COM2.
  Can't access COM4.

- Third Party Comm Driver Conflicts: Specialty communications devices with more than four ports or intelligent hardware replace Windows' standard COMM.DRV. Also, driver replacements are sometimes used by ISVs to boost performance or add extended features. Replacing the standard Windows driver leads to compatibility problems, since this driver might include obsolete code for standard COM and LPT ports.

  To address this problem, Chicago's communications API and DDI should be modular and extensible to allow IHVs and ISVs to add the code they need without replacing Windows' original support code.

9.3.2. Using Modems

Connecting to remote computer via modem is a several step process which exposes the user to a number of complicated interfaces before a connection is actually established.

- Selecting the Correct Port: Which port is my printer connected to? My mouse? What port is my internal modem supposed to be? Which ones can I use at the same time? How do I configure COM5? These are the kinds of problems that users encounter at the very beginning of a communications session.

  When users want to print, they just want to specify a printer, not a "printer on COM1" or "printer on LPT1". Similarly, when they want to dial out, they just want to specify a modem (or none at all if there's only one!). And in all cases, they don't want to have to worry whether they can use two devices at the same time.

- Configuring the Port: It is very difficult for users to set the correct parameters to establish a connection between their communications application, modem, and host. Little of the hardware mechanics of serial communications has been abstracted away from the user. Before a connection can be established, the user must set a handful of parameters depending on the capabilities of the host and remote PCs: baud rate, data bits, stop bits, handshaking, parity, etc.

  Also, users do not understand why there are three places to configure serial ports: MS-DOS mode command, Control Panel, and in applications themselves.

- Configuring the Modem: The Hayes AT command set is by far the most common, but it is still a cryptic language users should be isolated from. Some modems support additional commands for extended features.

- Sharing One Modem With Multiple Applications: A single port may provide connections to a number of applications at different times. Although none of these applications may actually be using the port at a certain time, it may have ownership of the port, thus preventing access to the other applications. This is common with MS-DOS communications applications, and the reason for the "Device Contention" setting in the 386 Enhanced Control Panel. Also, an application might be awaiting a call on a modem, preventing another application from dialing out on the port.

9.3.3. Using Printers

Most users would not think of their printer as a "communications" device, but it is in the sense that it is accessed via the communications hardware and application interfaces.

- New Port Types Are Coming: New high-speed and bi-directional port types must be adequately supported by Chicago.

9.3.4. Using Electronic Mail

In the office, many users access e-mail services such as MCI Mail or CompuServe via modem. Users who access e-mail via network while in the office usually switch to a modem for remote access. In either case, background downloading of received messages and uploading of sent messages is used in order to conserve phone charges.

- Data Loss During Text Transfers: Using other applications while e-mail messages are transmitted in the background slows throughput, taking more time and increasing phone charges. Frequently, e-mail transmissions are simple text uploads and downloads, extremely susceptible to data loss due to system overhead. If anyone in the system disables interrupts for too long, a message can be garbled. This is likely to happen during background operation. In Windows 3.1, opening and closing MS-DOS VMs disables interrupts long enough to drop characters at 2400 baud.

9.3.5. Browsing Info Services, BBS's

In addition to local bulletin board services, CompuServe, America On-line, and Prodigy are becoming increasingly popular. CompuServe is now marketing to television audiences, an indicator of the movement of increasing number of PC users into modem communication activities.

When accessed with basic terminal emulators, communications demands for BBS's are not rigorous, since command line interfaces do not require high bandwidth character transmission. CompuServe is moving towards a graphical interface, like its competitors America On-line and Prodigy. Graphical interfaces send a greater amount of data across the wire, driving up the need for reliable high-speeds during browsing.

- Accessing On-line Services: Chicago should make it easy for customers to access popular on-line information services.

9.3.6. Transferring Files

Once the user has installed a modem, set port and modem parameters, dialed the modem, established a connection, and started browsing, they usually need to transfer files to/from the host. This is where most users encounter performance-related limitations of communications under Windows.

File transfers are usually performed in the background due to their long duration. Unlike e-mail text transmissions, binary file transfers are usually performed with error correcting protocols to accommodate line noise and dropped characters at the receiving end.

- Slow Throughput, Data Loss During File Transfers: Performed in the background, file transfers are slowed by CPU activity for foreground applications. This introduces delays servicing I/O buffers and protocols. Plus, retransmissions are required to correct errors caused by dropped characters at the port if interrupts are disabled for too long or time-outs occur due to scheduling latencies. Since file transfers often are long operations to begin with, reductions in throughput due to system overhead and errors adds significantly to download time. Also, if too many errors are encountered, a transfer will be aborted altogether.

- Optimizing File Transfers is Difficult: There are a number of tweaks a user can make in their software configuration in order to boost throughput and prevent errors during file transfers:

- Run background MS-DOS comm applications as an icon, not in a window
- Set "Lock Application Memory" in the PIF for MS-DOS comm apps.
- Set "Background Execution" in the PIF for MS-DOS comm apps.
- Change the 16550A FIFO trigger level (coming in Win3.1a)
- Increase the size of the VCD buffer using "COMxBuffer=..."

Unfortunately, these are complicated and not well understood. Users should be isolated from these settings as much as possible.

9.3.7. Using Terminal Emulators

TBD: How can terminal emulator selection and options be made easier?

9.3.8. Sending & Receiving Faxes

TBD: What DDI and API support is needed to integrate fax management?

9.3.9. Accessing Network Resources

TBD: Remote access products provide a way to access network resources via modem. Other than high-performance, what will Chicago need in order for this to be built-in?

9.3.10. Using Telephones

TBD: What will the Telephone API (TAPI) group need to talk to telephones in Chicago?

9.3.11. Using 3270 Emulators

TBD: Other than kernel improvements, how should 3270 support be improved?

9.3.12. Controlling Other Serial Devices

TBD: What other-devices are controlled on comm ports? What does Chicago need for these? For example, industrial protocols will be handled better by allowing ring 0 protocol handlers.

9.3.13. Writing Communications Applications

ISVs are used to writing low-level communications code to control devices, protocols, etc. They face a number of limitations in Windows.

- Determining Modem Status: The Windows 3.x APIs which provide modem and line status indicators are poorly documented and do not work correctly. Only an undocumented PSS workaround saves the day.

- Device Selection & Contention: There is no way for an application to determine what devices and ports are actually installed, and which are available for use. This leads to device contention warnings for the user, possible device problems (e.g., trying to use the mouse port), etc. An API is needed to allow applications to easily offer a list of available devices to the user.

- API Too Device Dependent: Applications must know low-level details about how to establish connections with devices. For example, they need to know modem command sets in order to dial, hang up, etc. Sending and receiving data and monitoring connection status is also device dependent. An generic API is needed to isolate applications from device specific parameters of communications channels. This is increasingly important as new, more complicated devices are introduced.

9.4. Features

9.4.1. Windows 3.1 Compatibility

Priority 1

Objective

Windows 3.1 communications applications and drivers will load and run in Chicago. Communications driver replacements currently in use include:

- TurboComm: This 3rd party driver from Bioengineering Research provides improved communications throughput for MS-DOS applications by virtualizing the 16550A UART FIFO. It will not be useful under Chicago, which will do the same for MS-DOS VMs.

- Remote access software: Some s/w packages like CarbonCopy may replace COMM.DRV or VCD. These applications will be identified and kept compatible with reasonable efforts. We must cooperate with these ISVs to make this happen.

- Hayes ESP: Hayes' high-speed DMA serial port.

- Multi-port adapters: Communications port devices with more than COM1-COM4

- Intelligent adapters: Communications port devices with on-board transmission h/w.

- Misc. OEM: Specialized application software and devices may have modified the communications driver from the DDK for specific support. In some cases, these may be unnecessary under Chicago.

Feature Description

Version 3.x communications drivers should replace Chicago's built-in support for local COM ports. Some new communications features may not be available, however.

The undocumented procedure for accessing modem status information from an internal data structure must be preserved! PSS has provided this to major comm ISVs for Windows 3.x products.

Status

DavidThi's Cougar communications spec was devised with Win3.1 compatibility as a design goal.

Open Issues

9.4.2. Solve Port Address/IRQ Problems

Priority 1

Objective

Chicago must solve serious problems experienced by users attempting to configure serial port address and IRQs. Windows 3.1 assigns default serial port identifiers (e.g. COMx), addresses, and IRQs according to an interpretation of the IBM BIOS spec that is not used by customers, OEMs and modem vendors. This is a severe problem causing a very large number of reported serial device access problems, including mice, modems, and COM ports.

Feature Description

Chicago's initialization of COM ports should be based on a fix developed for Windows 3.1a which causes the COMM.DRV and VCD to handle BIOS data table addresses and SYSTEM.INI entries differently. See Appendix A, "Port Addressing and IRQ Problems", for a detailed description of this problem and its symptoms.

Status

The COMM.DRV update for WDL/Win3.1a has been developed, but not yet beta tested.

Open Issues

- How should Chicago determine address/IRQ for UART ports present but non-standard or not listed in the BIOS data table? If possible, the user should not have to know this information.

9.4.3. 16550A UART FIFO Support

Priority 1

Objective

Chicago will provide greater robustness and performance at high baud rates for MS-DOS communications applications using local serial ports with 16550A compatible UARTs. This is the single biggest sore spot that communications power-users have with Windows at present, and causes many to purchase a comm driver replacement package. By the Chicago time frame, 16550A devices will be more prevalent, and not only power-users will require this feature, so this is a must.

Feature Description

Local COM ports that utilize the 16550A style UART contain a 16 byte FIFO buffer to prevent character overflow due to interrupt latency, and reduce interrupt overhead overall. In Chicago, the 16550A FIFO will be fully utilized as follows:

- FIFO Enabling: The FIFO should be enabled whenever a Windows or MS-DOS application is accessing a local 16550A UART port. (The Win3.1 Virtual Comm Device (VCD) does not enable the 16550A FIFO for MS-DOS comm applications.) If COMBUFF if virtualizing access to the UART, the FIFO should be used for MS-DOS comm applications regardless of whether the application attempted to detect and use the FIFO.

- FIFO Virtualization: A 16550A FIFO should be virtualized for 16550A-aware MS-DOS comm applications regardless of the physical UART present. This reduces overhead of simulating character interrupts into non-Windows VMs, and has been shown to boost throughput by up to 20% for FIFO-aware applications running on non-FIFO UARTs.

- FIFO Trigger Level: Default FIFO trigger setting should be 8, and be configurable from SYSTEM.INI. (NOTE: Windows 3.1 hardcodes the trigger level to 14.)

Status

MX 2120526
CONFIDENTIAL

An updated driver for the WDL and Win3.1 uses a default trigger level of 8 and provides a new SYSTEM.INI switch to change this setting.

Open Issues

9.4.4. Kernel Improvements

Priority 1

Objective

The Cougar kernel and device layer must be "comm friendly", such that high-speed communications are not interfered with during device access, paging, etc.

Feature Description

Interrupts must not be disabled for periods long enough to cause overruns on serial ports.

The VM shutdown bug which disables interrupts for too long should be fixed.

Extensive tests focused on communications performance should be integrated into Cougar validation procedures.

Status

Open Issues

9.4.5. Improved Device Contention

Priority 1

Objective

Chicago should reduce problems encountered by users who attempt to use a device already in use by another MS-DOS or Windows application, or an available device on the same IRQ. Also, the device contention user interface in the Control Panel should be simplified.

Feature Description

Chicago should automatically share IRQs for devices which support this feature. The user should not have to put "COMIRQSharing=TRUE" in SYSTEM.INI for this to take effect.

A warning message should be displayed when an application attempts to open a port on a non-shareable IRQ which conflicts with another open port. The user should be allowed to cancel or continue, with the option of not displaying the warning message in the future.

Mouse drivers should claim the serial port they are using, if any.

Printer drivers should claim the serial port they are using, if any.

Status

Open Issues

9.4.6. Installable Port Drivers

Priority 1

Objective

The Windows 3.x COMM.DRV module is a monolithic driver supporting both serial COM ports (COM1-COM4 standard) and parallel LPT ports (LPT1-LPT4 standard). This binding restricts compatibility and expansion to support non-standard and multi-port devices. To solve this problem, Chicago should allow any number of port device drivers to be installed in order to provide robust support for the variety of communications devices that can be installed into a PC.

Feature Description

Chicago should provide a port device driver interface and API with the following features:

* Multiple Installable Port Drivers: Any number of port device drivers can be installed and loaded. This may include UART COM ports, LPT ports, intelligent adapter ports, etc.

* Multiple Ports per Driver: A port driver can register itself at Windows startup as one or more devices, supplying a name, type, base address, IRQ, and other parameters.

* Port Names: Port identifiers should not be restricted to COMx format, e.g. COM1, COM2, etc. Rather, ports should register with meaningful user names, e.g.:

  Local Serial Port 1
  Local Serial Port 2
  Local Parallel Port 1
  Local Parallel Port 2
  Hayes Enhanced Serial Port
  Hayes v.9600 Modem

  These names will be returned by a port enumeration API/UI. Applications can open a port by this name, rather than "COM1", etc.

* Dynamic Loading: Port drivers should be loaded at init time to register, but not consume system resources until opened. Drivers for closed port devices should be unloaded automatically.

* Extended Functions: A generic escape/IOCTL function should allow device specific functions to be called by applications.

* Configuration Dialog: Port drivers should supply an appropriate dialog to allow the user to change device-specific parameters. A Windows API should provide access to this dialog for applications.

* Standard COM Port Driver: Chicago should support at least eight 8250/16450/16550 compatible UART based serial ports. COM1 through COM4 should default to standard address/IRQ values. COM5 and up should require the user to set these values

* Standard LPT Port Driver: Chicago should support at least four standard LPT ports.

Status

DavidTh's Cougar comm specification examines implementation of port drivers.

Open Issues

* Should port drivers be DLLs or Cougar VxDs? If VxDs, how will configuration dialogs be displayed?

* Should comm be integrated into Dragon? Why? What about config dialogs?

* How should ports > COM4 be accessed? Specifying addresses/IRQs is counter to Plug and Play objectives.

* Should portions of the Win/NT Service Provider Interface (SPI) be used for Chicago port drivers?

* What other port drivers should go in the box? Possibilities include Hayes ESP, BAPI, Novell NASI, X.25 with X3.PAD, DEC LAT

9.4.7. INT 14h API Support

Priority 1
Objective

HIGHLY CONFIDENTIAL

MX 2120528
CONFIDENTIAL

CMS 00013822

RBC 04223

Compatibility with MS-DOS communications applications that use the Int14h API for communications rather than accessing the hardware directly. Usually, Int14h is used in conjunction with redirector TSRs to access remote serial ports and modems.

Feature Description

INT 14h is a BIOS API supported by some MS-DOS communications applications. In MS-DOS communication applications, users select Int14h modes instead of communicating directly with the COM port so that other loaded software can redirect the I/O to remote devices. Chicago should provide an Int14h interface for MS-DOS applications. This interface will redirect I/O to the correct Chicago port driver.

Status

Open Issues

9.4.8. Bi-Directional ("Bi-di") Parallel Ports

Priority 1

Objective

Chicago should provide port drivers for new parallel port devices, including "Zippy" and "Boise".

Feature Description

Individual port drivers should be written.

Status

Open Issues

- Can WPG do this?

9.4.9. Modem Status Information API: GetCommEvent()

Priority 1

Objective

Windows 3.x does not meet the needs of comm applications which must monitor the status of modem status lines. The Windows 3.0 functions SetCommEventMask and GetCommEventMask were designed to provide this functionality, but are buggy and poorly documented. Unfortunately, compatibility constraints require that these APIs not be altered. Most ISVs are using an undocumented procedure provided by PSS which allows this information to be accessed from an internal data structure maintained by the comm driver.

Feature Description

A new GetCommEvent function should added to Win16 to supply modem status information. This should be nearly equivalent to the Win32 GetCommEvent API.

NOTE: The undocumented procedure for accessing modem status information from an internal data structure must be preserved! PSS has provided this to major comm ISVs for Windows 3.x products.

Status

LesterW examined this in '91 for the Win32 comm API, Chicago will leverage this work by confirming with ISVs that it is correct & sufficient.

DavidThi has examined modem status information in his Cougar port driver spec.

Open Issues

- How closely can/will Chicago match Win32's implementation?

9.4.10. Plug & Play

Priority 1

Objective

Chicago should perform as much device setup as possible in order to allow the user to access communications devices without having to specify which port they are on. For example, they should be able to simply print to a printer by name, and accessing a modem should be automatic if there is only one plugged into the PC.

Feature Description

As many devices as possible should be automatically assigned to port IDs using intelligent detection, without requiring the user to state which device is "on" which port. Serial and parallel ports can be scanned at Setup time in order to determine what's out there.

Device selection can be made easier by using friendly device product names instead of port abbreviations.

A default modem should be specified similar to default printer so that dialing out does not require the user to specify a serial port.

Status

Open Issues

9.4.11. Using Remote Communications Devices

Priority 1

Objective

Users should be able to use remote communications devices, such as network modems, etc. Remote devices should be available from all existing communications applications, not just Chicago applications. For example, a remote port driver might be loaded and assigned as COM5.

Feature Description

A port driver should register remote ports. Applications will be able to open and access remote ports by name.

Status

Needs investigation.

Datastorm is preparing input on this topic. Greenleaf has offered assistance.

Shiva should also be contacted due to their Net/Modem products.

Other remote comm product vendors include Microtest and Artisoft.

Open Issues

- How can/should Chicago users access non-Chicago remote communications devices?
- Can existing MS-DOS and Windows comm applications access ports beyond COM4? If not, then an "aliasing" feature is required so that users can specify COM1-COM4 identifiers for non-local ports.
- How much does Int14h support buy? Will it work with existing Int14h redirectors? Does it make sense to use an Int14h port driver to be redirected by another TSR or VxD? Or, should just a redirector VxD be provided?
- Should a Novell NASI driver be provided? Could/would Novell do this?

9.4.12. Sharing Local Communications Devices

Priority 1

Objective

Users should be able to share their local communications devices with others on the network.

Feature Description

Sharing local communications devices would have a similar API as sharing a file directory or printer.

Status

Needs investigation.

Open Issues

- Should Chicago users be able to share local communications devices with non-Chicago users on the network?
- Can protocols be shared? Protocols should run as local to the physical communications device as possible. The comm redirector should be able to register non-local protocols. For example, a comm redirector might run on a Chicago server. The redirector would publish local port devices plus the protocols available for these devices. On the client, the remote port driver registers the protocols received by the redirector as local protocols available only on the remote device.

9.4.13. Protocol API & Drivers

Priority 1

Objective

Throughput and reliability of communications protocols can be improved by moving protocol code into ring 0. Communications ISVs (including Wonderware) report that Windows is unsuitable for real-time protocols due to scheduling latencies. That is, although incoming characters are received fast enough from serial ports by Windows, receiving applications are not scheduled quickly enough to service the protocol correctly.

Also, communications application development can be made easier by having protocols to plug-in separately. This would allow a simple comm application written to the Chicago API to use any installed protocol, without requiring special built-in handlers.

Feature Description

Chicago should allow applications to install protocol handlers which run at interrupt time. This will avoid time-outs, reducing the number of re transmissions required. Plus, protocol turnaround will be faster; so data flow will not be halted as often in order to wait for the ring 3 application to ACK the protocol. Features of ring 0 protocol drivers include:

- Installable: ISVs and IHVs should be able to write and install their own protocol handlers appropriate to their application and hardware. We can expect some ISVs to market installable protocols directly to other ISVs. Other ISVs may move code that now exists in their applications into a protocol driver.
- Dynamic Loading: Protocol drivers should be dynamically loadable by applications.
- Multiple Protocols: Multiple protocols can be applied to an open communications channel.
- Simple Protocol API: The Windows comm API should be extended to allow easy access to protocol handlers. Comm applications should be able to simply service I/O data streams after invoking a protocol handler. A generic API should allow any application to make use of any installed protocol.
- Configuration Dialog: Protocol drivers should include their own configuration dialog to allow the user to customize protocol parameters.
- Common Protocols: Chicago should include a basic set of common protocols. This includes standard flow control (XON/XOFF) for text, hardware flow control (RTS/CTS), and possibly some common binary protocols, such as XModem and ZModem.

Open Issues

- Must protocol drivers be ring 0? Are there protocols that don't require ring 0? If so, would these run better at ring 3? I.e., does protocol processing at ring 0 introduce unnecessary system overhead?
- Are protocol drivers VxDs or DLLs? If VxDs, how do they display a configuration dialog:

9.4.14. Fax Management

Priority 1

Objective

Chicago should provide integrated fax management, like printing.

Feature Description

Status

Open Issues

- Move this to the Utilities group?

9.4.15. Modem API & Drivers

Priority 2

Objective

Chicago should isolate users from cryptic modem configuration options and command languages. A simple API should allow applications to make connections in a modem-independent fashion. The user and application will merely have to supply the phone number to dial; applications may provide a phonebook feature for this.

Feature Description

Modem APIs should be provided for the following functions:

- Dial: Causes the modem to dial the specified telephone number. This should return immediately, and status messages should be sent to the application via window message or call-back.
- Answer: Causes the modem to answer a ring.
- Auto-Answer: Causes the modem to enter auto-answer mode. A message should be sent to the application via window message or call-back.
- Hang Up: Causes the modem to hang up the current line.
- Setup Dialog: Causes the modem driver to display a dialog to allow the user to set modem-specific options.
- Set/Query Settings: Modem options for tone/pulse, speaker on/off/volume, dial, carrier, modulation times, etc. can be queried and set.

Modem drivers should be DLLs capable of supporting the modem API, and also meet the following requirements:

- Dynamically Loadable

Status

DavidThi has examined modem driver requirements in his Cougar comm specification.

MX 2120532
CONFIDENTIAL

Open Issues

- Does this accommodate fax modems?
- Should apps have to use a modem API at all? Could this be abstracted away from the application as well as user? I.e., an OpenComm() on a modem port might automatically dial a pre-determined phone number set in the modem setup dialog.

9.4.16. Communications Manager

Priority 2

Objective

A central user interface location for the installation, configuration, and sharing of installed communication devices. Eliminate user confusion about why there are port settings MS-DOS, Control Panel, and applications themselves.

Feature Description

The Communications Manager should be a counterpart application to the Print Manager, with similar functionality. The Communications Manager should perform the following functions:

- Installation and Configuration of Communications Devices
- Sharing of Local Communications Devices
- Connection to Remote Communications Devices
- Show Status of Installed Communications Devices

See Appendix C, "Communications Manager", for notes about the design of this accessory.

Status

Needs approval and design.

Open Issues

9.4.17. Communications Setup Dialog

Priority 2

Objective

Chicago should provide a common dialog to make it easy for ISVs and users to establish connections with remote computers.

Feature Description

The communications setup dialog should show a list of installed and available local and remote communications resources, including serial and parallel ports, network devices, modems, fax modems, etc.

The Printer control panel should use the comm setup dialog for assigning a printer to a port.

The comm dialog should provide network connection services similar to the Printer "Connect..." button.

The comm dialog should include a "Setup..." button which displays a port-driver specific configuration dialog.

Status

Needs investigation.

Open Issues

9.4.18. Auto-Negotiate Protocol

Priority 3

Objective

Chicago could introduce a connection protocol used to transparently and intelligently establish connections and determine transfer protocols for the user when a connection is attempted. This would isolate the user from complicated parameters required in order to make a connection to a remote computer, including:

Baud Rate: 2400, 4800, 9600, 19200, etc.
Data Bits: 5, 6, 7, 8
Stop Bits: 1, 1.5, 2

Parity: None, Odd, Even, Mark, Space
Flow control: XON/XOFF, Hardware, None

Also, binary file transfers require users to specify a protocol, e g. ASCII, Kermit, XModem, YModem, ZModem, etc. Each of these protocols is slightly different from the other, and some are better than others depending on connection variables, such as baud rate, line noise, system overhead, data type, etc.

Feature Description

When a port is opened, Chicago will attempt to establish a connection with the remote computer via a universal protocol language. If the other end is listening for this protocol, the two will negotiate to match up the fastest common denominators for baud rate and other connection parameters. Also, the best available binary transfer protocol available will be selected, and applied if the user initiates a file transfer.

Status

Needs much investigation.

Open Issues

- Do similar protocols already exist? If so, can they be adopted?

9.4.19. Power Management

Priority 3

Objective

Chicago should provide a device driver interface to allow better power management of communications devices.

Feature Description

Status

Needs investigation.

Open Issues

9.5. Resource Requirements

Manpower and material resource requirements to achieve Chicago's communications objectives are discussed in this section.

9.5.1. Laboratory

Chicago should emphasize the testing of real world communications usage. In order to do so, the following test lab resources should be set up for the duration of development and testing:

- At least four networked 386/20's should be available for communications testing near analog phone lines, etc. A high-speed 486 should be available for comparing Chicago's top-end capabilities with OS/2.

- Various permutations of on-board ports, port adapters, and modems. The permutations are important! Windows 3.x users have problems unique to certain combinations, including:

  COM1 only
  COM1, COM2, mouse on COM1
  COM1, COM4
  COM1, COM2, COM4, mouse on COM1
  COM1, COM2, COM4, mouse on COM2
  COM1, COM2, COM3, COM4

MX 2120534
CONFIDENTIAL

- UARTs. On-board and adapter ports and modems using these UARTs:

8250
16450
16550
16550A

- Modems

  Internal adapters
  External modems
  1,200 through 38,400 baud
  Hayes compatible
  Non-Hayes compatible (TBD)
  Wireless (if available)

- Four analog telephone lines (at least). Used to test multiple simultaneous communications sessions.

- On-line information service accounts:

  CompuServe
  Prodigy
  MCI Mail

- MS-DOS application software:

  ProComm Plus for MS-DOS
  CompuServe Information Manager for MS-DOS
  Prodigy
  TapCIS
  Fax applications, top 2, e.g. Delrina
  Remote access packages, top 2, e.g. Norton pcAnywhere, Carbon Copy

- Windows application software:

  ProComm Plus for Windows
  CompuServe Information Manager for Windows
  Dynacomm
  Fax packages, top 2, e.g. Delrina WinFax, etc. (unless obsolete!)
  Remote access packages, top 3, e.g. Norton pcAnywhere, Carbon Copy

9.5.2. Development

Microsoft has a reputation for paying lip-service to communications services in MS-DOS and Windows, and this will change after Chicago! In order to achieve the Chicago communications objectives, development attention on the order of that applied to printing for Windows 3.1 is advised.

The Chicago development team should have at least one engineer who has significant experience writing communications applications and device drivers. Most of the Chicago features are not new to the industry, so there is no need for Chicago to reinvent the wheel when it comes to API and DDI design. For this reason, a comm veteran is a good bet for getting design & code implemented in an efficient fashion.

Chicago's communications features can be categorized into DDI, API, and UI as follows:

Device Driver Interface (DDI)
Windows 3.1 Compatibility
Solve Port Address/IRQ Problems
16550A UART FIFO Support
Kernel Improvements

Installable Port Drivers
Bi-directional Parallel Ports
INT 14h Support
Using Remote Communications Devices
Sharing Local Communications Devices
Power Management

Application Programming Interface (API)
Modem Status Information API: GetCommEvent()
Improved Device Contention
Modem API & Drivers
Protocol API & Drivers
Auto-Negotiate Protocol

User Interface
Plug & Play
Fax Management
Communications Manager
Communications Setup Dialog
On-Line Information Access

9.5.3. Testing

One of the reasons that communications problems lurking in Windows 3.1 were not caught until late in the game is that common end-user communications scenarios were not being tested throughout the development cycle. Rather, a few PCs were directly connected, and top baud rate tested in the foreground and background.

The following testing objectives should be fulfilled in order to meet the Chicago communications objectives.

- Expert Communications Knowledge: Test engineers and developers must become experts in the needs and behaviors of Chicago communications customers. This includes knowledge of common operations, popular hardware and software, and optimizations. This should be much like how printing and networks were tested for Windows 3.1 and Sparta.

- Performance/Robustness Benchmarks: We currently have no good way to benchmark comm performance and reliability. A suite of automated tests should be developed that allow us to compare against OS/2, MS-DOS applications, different modems, baud rates, etc. These should be run in common end-user configurations, using phone lines instead of direct connections, etc.

- Performance Monitoring: Changes in other areas of the OS, especially kernel and devices, can affect communications performance and robustness. There must be continual verification that Cougar and other changes do not cause communications problems as they are checked in. This occurred due to FastDisk and other system performance optimizations in Windows 3.1.

- Emphasis on Background Communications: Customers use Windows because it allows them to multi-task, and communications is one area where this feature is frequently applied. Usually, users do not want to sit and wait for files to download, they'd rather be doing other work. Our tests should rigorously test background communications, with ample amounts of foreground application and device activities.

- Broad Configuration Testing: Many of the product support problems encountered stem from inadequate testing of the huge variety of communications hardware configurations that exist. In particular, Windows 3.1 customers are having problems when they have "blank spaces" in their COM ports, e.g. no COM1/COM4, no COM2/COM3. Many of these combinations went untested for Windows 3.1, simply due to focus on performance testing on standard COM1/COM2 configurations.

  A reasonable way of testing a variety of common hardware combinations should be determined. One way is to keep internal testers close to reports from beta testers, so we can have a good understanding of what our customers are attempting to do, and have the flexibility to easily try these out.

9.5.4. Program Management

Program Management tasks required to achieve the Chicago communications objectives are as follows:

- Determine Requirements of Microsoft Groups: Keep these teams in the loop for feature & design reviews:

  Windows NT (LesterW, TonyE)
  ISV Relations (???)
  Hardware Vendor Relations (???)
  PSS Developer Support (BryanW, et al)
  Telephone API (CharlesFi)
  Mobile Windows (SteveL)
  Modular Windows (TomasM)
  MS Mail (???)

- Determine ISV Requirements: Gotta have a design review organized by Systems Marketing. Important ISVs include:

  Datastorm (ProComm/Plus, ProComm/Win)
  Wonderware
  Shiva (NetModem/E)
  Digital Communications Associates (Crosstalk)
  FutureSoft (DynaComm)

- Design Specification: The usual.

- Obtain Third Party Drivers: If Chicago is to include support for non-standard devices:

  Windows Printing Group (Zippy)
  Hewlett Packard (Boise)
  Hayes ESI
  Novell NASI

- Get Great Beta Testing: Better beta tester monitoring would have given us earlier warning of problems caused by kernel changes during Windows 3.1. Program Management will build a list of power-comm users. These aren't hard to find, they are very vocal on CompuServe forums.

- Usability Testing?: This would be a significant expense. Would it be worthwhile to determine if Chicago actually makes communications easier?

9.5.5. User Education

Communications expertise here would be great to help devise Wizards and simple ways of explaining tough communications details.

More TBD.

10. Display Support

This section discusses standard and third party display support in the Chicago Windows release.

10.1. Overview

10.1.1. The 5 Chicago Improvements

Chicago's primary improvement areas apply to display features as follows:

- Plug & Play: Chicago will allow many more users to automatically access Super VGA resolutions and colors without having to identify adapter type and brand, insert floppies, obtain third party drivers, etc.

* New User Interface: A generic, intuitive Display Control Panel will allow users to configure colors, resolution, and other display related parameters.. Changing drivers will happen in Windows, not MS-DOS.

* Integrated and Complete Protect Mode System: A 386

and 486 optimized DIB engine and page fault handler will provide fast, robust drawing code for Super VGA and frame buffer displays.

* Great Network Client and Peer Server: Not applicable.

* 32-bit Application Support: Not applicable.

10.1.2. The 5 Chicago Requirements

Chicago's primary technical requirements apply to display features as follows:

<u>* Compatibility: Windows 3.0/3.1 display drivers, grabbers, and VDDs will be compatible with Chicago.</u>

<u>* Great 4 Mb System</u>

<u>* Performance: All Chicago display drivers will be use the fastest drawing code available, including the high-speed GDI DIB engine.</u>

* <u>Robustness: A Universal Display Driver will eliminate buggy code from third party drivers by using the GDI DIB engine.</u>

* <u>On Schedule: Of course.</u>

10.1.3. From the CEO...

Here's what BillG has to say about what Chicago display support should be:

From:	Bill Gates
Date:	May 6, 1992
Subject:	Windows 1993

8-bit video drivers should be shipped for all popular machines. We should make sure we ship the fastest video drivers available. We should pick a few accelerator chips/designs for fast video cards and decide to make them mainstream by having great drivers we support and pushing oems to use one of those approaches.

I think there should be an equivalent of UNIDRV for video drivers. A lot of the code is common. Someone good has to look at what code sharing can be done between video drivers. I'm sure a lot of the code is common. Going to a UNIDRV approach for displays will free up some disk space.

10.1.4. Non-Objectives

<u>* NT Display Model: Chicago will not support Windows NT display drivers. The device driver model will be similar, but not identical.</u>

10.1.5. *Important Features*

The important Chicago features for displays described in this specification include:

<u>* Windows 3.1 Compatibility: Old drivers run under Chicago.</u>

<u>* Universal Display Driver: Built-in EGA, VGA, plus easy mini-driver interface.</u>

* Generic Super VGA Mini-Driver: Support for 90% of Super VGA adapters.
* Display Memory Page Fault VxD: Part of Universal Display Driver.
* GDI DIB Engine Drivers: Small, robust, high-performance linear frame buffer drivers.
* Display Control Panel: Centralized & easy display configuration & driver installation.

* Easier Driver Installation: Isolates user from technical details.
* Fall-Back Display Mode: Windows runs even if the display driver fails.
* Universal VDD: Built-in EGA, VGA, plus mini-VDDs for extended hardware.
* Standard Grabbers: Built-in high-performance grabbers.
* Device Independent Color: Ability for drivers to match to printer colors.

* <u>Pen Compatibility: Drivers incorporate Windows for Pen features.</u>

Features under consideration that may also be developed:

* <u>Dynamic Screen Resolution: No need to restart after changing display resolution.</u>
* <u>Dynamic Color Depth: No need to restart after changing display colors.</u>
* <u>Physical/Scalable WYSIWYG: User scalability of display contents.</u>

MX 2120545
CONFIDENTIAL

HIGHLY CONFIDENTIAL

CMS 00013839

RBC 04240

* Monitor Drivers: Files used to optimize display modes for monitors.

10.1.6. Major Open Issues

* XGA: IBM Boca Raton owns development of this driver. Will we be able to get a Chicago driver without disclosure and other problems? Should we bring this in house?

10.1.7. Related Documents

* GDI Specification
* Device Independent Color Memo (DavidW)

10.1.8. Revision History

9/25/92 Original version distributed for review.

9/29/92 Removed "386+ Only Drivers" section, incorporated info into "Driver Summary".

Added "Important Features" section to "Executive Summary"

Moved "Driver Summary" to follow Features section

Reformatted for inclusion in Chicago spec

10.1.9. Terminology

BPP	Bits Per Pixel; 4 bpp = 16 colors, 8 bpp = 256 colors, 15 bpp = 32K colors, 16 bpp = 64K colors, 24bpp = 16M colors ("true color")
True Color	24 bits per pixel
VDD	Virtual Display Device; A virtual device (VxD) required to virtualize the display hardware.
Grabber	A DLL of extension 3GR which paints MS-DOS application windows. Only h/w specific if an IHV wishes to support proprietary display modes.
IHV	Independent Hardware Vendor; manufacturer of display chipsets or adapters; may be an OEM.
Flat model	A display type that can map video memory in a contiguous range of extended memory address space. Windows will draw to the screen by treating this memory range as a large DIB.
DIB	Device Independent Bitmap; a Windows data structure for images, the same format for all devices. See the Windows SDK for more details.

DDI Device Driver Interface; device independent software interface between device drivers and Windows core modules, such as GDI and USER.

10.2. Features

10.2.1. Windows 3.1 Compatibility

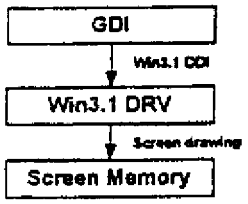
Priority 1

Objective

Windows 3.1 display drivers will load and run under Chicago, and GDI will simulate any features supported with new DDI.

Feature Description

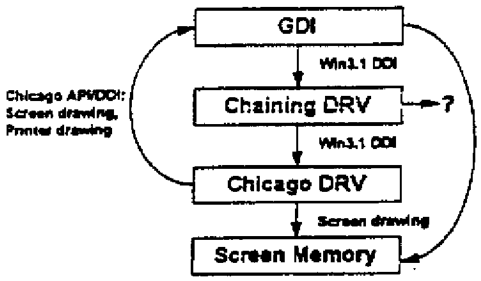
The calling sequence will be identical to Windows 3.1:



Some software products "chain" themselves in front of the Windows display driver. For example, remote control packages such as Carbon Copy do this to redirect graphics calls to the remote PC. Also, Bitstream's Facelift installs in a replacement display driver, which loads the existing display driver itself.

In order to remain compatible with these types of products, Chicago display drivers must fully support the full Windows 3.1 DDI in order to load and run "beneath" a chaining display driver. Chicago mini-drivers will be compatible, since the Universal Display Driver will be the module loaded by the chaining driver.

The following diagram shows the calling sequence of a chaining driver using the Windows 3.1 DDI of a Chicago display driver:



MX 2120549
CONFIDENTIAL

10.2.2. Universal Display Driver

Priority 1

Objective

Fast, robust, small, easy-to-write Super VGA and flat frame buffer display drivers. This will apply to the vast majority of Chicago displays.

Feature Description

DISPLAY.DLL will be a Csicago display driver module, i.e. it exports all of the standard Chicago display functions. It provides the followin; features for Chicago display support:

* Built in EGA and VGA/ DISPLAY.DLL will be installed on all EGA and VGA compatible PCs, even if Setup or the user selects a displad type and corresponding driver that does not use DISPLAY.DLL. This will provide a fall-back for Windows in the event that the user selects a display mode that is not supported by their display adapter or monitor.

* Uses the GDI DIB Engi le: 1/8/16/24 bit packed pixel display modes will be supported using the GDI DIB engine for high-performance, robust screen drawing. The DIB engine is also used for memory bitmap drawing. 16 color drawing code will be built into DISPLAY.DLL, since the GDI DIB engine will not support these 4-bit planar modes.

* Mini Driver Interface: small, simple mini-drivers will provide display modes other than EGA and standard VGA. These will use the DRVafile extension, but not export the same functions as a standard Windows 3.1 or Chicago driver. Mini-drivers wik be called to supply the following functions:

1) Initialization: Setting th e desired display mode. (Required)

2) Bank Switching: SuperVGAs that are not flat frame buffers will need to use the page fault VxD in order to perform bank switching

3) BitBLT Some adapters provide ways to accelerate BitBLTs, including latches and BLT engines. If this function is not supported, DISPL AY.DLL will perform all BLTs.

4) Hardware Cursors: DISPLAY.DLL will draw the cursor if the mini-driver does not handle the cursor itself.

<u>Additional functions (line-drawing, etc.) may be added to the mini-driver interface, as necessary. In general, however, non-frame buffer displays, or those with many extended features or</u>

<u>acceleration should not use mini-drivers. For example, mini-drivers are not appropriate for 8514/A or TIGA displays.</u>

The Universal Display Driver will be loaded under the following two circumstances:

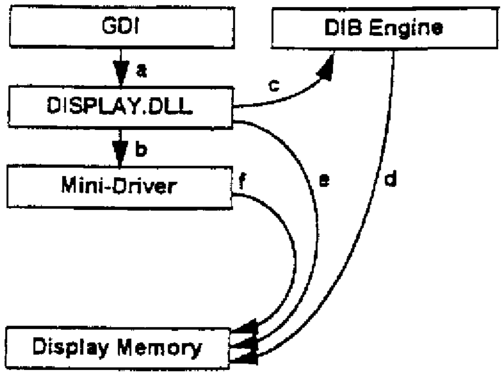
- Mini-Driver Installed: SYSTEM.INI will contain the following:

```
[boot]
display.drv=display.dll
```

<u>DISPLAY.INI will specify the mini-driver to be used by DISPLAY.DLL, plus screen resolution, colors, and other parameters.</u>

- Fall Back Mode: The display driver specified by SYSTEM.INI failed to load. DISPLAY.DLL will be used instead, falling back to EGA or VGA as appropriate.

The calling and drawing sequence of the Universal Display Driver is illustrated below:



a) GDI loads DISPLAY.DLL as the primary display driver. If no mini-driver is installed, the display is run as EGA or VGA.

b) DISPLAY.DLL loads the mini-driver specified by DISPLAY.INI, calling it's initialization function. The mini-driver sets the display mode specified by it's own private INI setting.

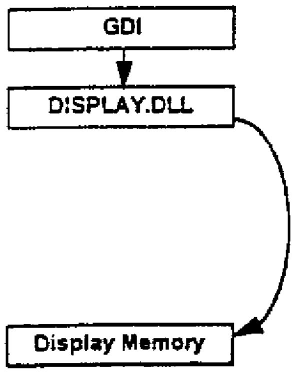
c) DISPLAY.DLL calls the DIB engine for drawing of 1/8/16/24 bit packed pixel display modes.

d) The DIB engine draws directly onto display memory.

e) DISPLAY.DLL may draw directly onto display memory for 4-bit planar (16 color VGA) displays.

f) The mini-driver may draw directly onto display memory for acceleration or cursor.

If no mini-driver is present, the picture is much simpler, since DISPLAY.DLL does all of the drawing for EGA and VGA 16-color display modes:



Status
DISPLAY.DLL will be easy to write, a merger of EGA.DRV, VGA.DRV, and calls to the GDI DIB engine. Development should design the mini-driver interface next.

Open Issues

10.2.3. Display Memory Page Fault VxD

Priority 1

Objective

Allow non-flat frame buffer display drivers to take advantage of the GDI DIB engine in order to be faster, smaller, more reliable, easier to write and maintain. This applies to bank-switched VGA chipsets, not "modal" or "fixed function" displays such as 8514/A, XGA, S3, ATI Ultra, etc.

Feature Description

Super VGA chipsets access display memory beyond 256K through a technique called "bank switching". Windows 3.0 and 3.1 display drivers for these chipsets were coded so that all graphics operations performed constant checking to determine when a 64K memory boundary is crossed, and then calls hardware dependent code within the driver to cause the chipset to reset for the required memory bank. Unfortunately, this architecture has many disadvantages, resulting in drivers that are:

Bigger, due to duplicate h/w independent code in drivers for the variety of chipsets on the market.

Buggier, due to more complex code to handle bank-switching. IHVs have a very difficult time keeping up to date with changes and improvements in the Windows DDI.

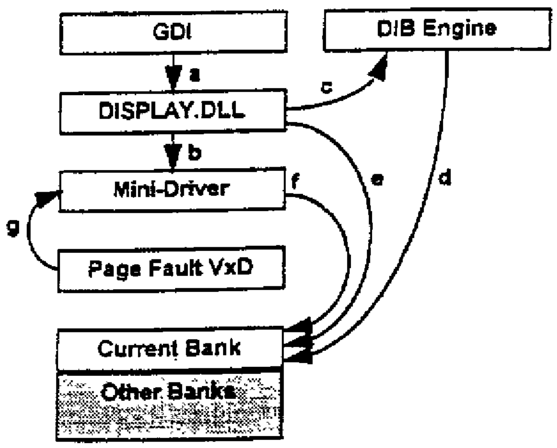
Slower, due to constant bank-switch conditionals.

A solution is needed to the problems above, and to allow IHVs to write the smallest amount of h/w dependent code to get Windows running on their chipset, and then focus on optimization for specific hardware features.

Chicago is being optimized for fast drawing on flat model DIB devices at 8bpp and higher. This code can be leveraged for non-flat model displays in order to address the problems inherent in bank-switching Super VGA drivers.

A display mini-driver for bank-switched display hardware can be based on the new Chicago flat model driver which calls GDI for all graphics rendering on a flat DIB surface. However, when screen memory is bank-switched rather than completely accessible in extended memory address space, a page fault handler VxD will be installed to catch occasions when GDI writes out of the current screen memory bank. The display mini-driver will be responsible for handling the bank-switch, and then GDI will continue writing as if the fault never occurred.

The following diagram and steps explain how this works:



MX 2120555
CONFIDENTIAL

HIGHLY CONFIDENTIAL

CMS 00013849

RBC 04250

a) GDI loads DISPLAY.DLL as the primary display driver.

b) DISPLAY.DLL loads the mini-driver specified by DISPLAY.INI, calling its initialization function. The mini-driver establishes a virtual flat DIB display surface by calling a page fault DDI provided by the page fault VxD.

c) DISPLAY.DLL calls the DIB engine for drawing of 1/8/16/24 bit packed pixel display modes.

d) The DIB engine draws directly onto the virtual display.

e) DISPLAY.DLL draws directly onto the virtual display for 16 color modes.

f) The mini-driver may draw directly onto display memory for acceleration.

g) When the mini-driver or DIB engine attempts to write to a display memory location that is not within the current bank of the display hardware, a page fault occurs. The page fault VxD notifies the mini-driver, which performs the required bank switch.

10.2.4. Super VGA Mini-Driver

Priority 1

Objective

Chicago should have built-in support for at least 90% of Super VGA adapters. Most OEMs are shipping on-board VGA's with at least 512K of video memory, supporting 640x480 in 256 colors, and 800x600 and 1024x768 in 16 colors. (1Mb of video memory allows 256 colors at 800x600 and 1024x768.) Unfortunately, most customers cannot identify what brand of chipset they have or how much display memory is present, and thus cannot identify the correct driver to use to get colors and resolutions beyond standard VGA.

Feature Description

Although generic support for 100% of adapters is difficult due to varying chipset requirements, Chicago should include a Super VGA mini-driver (SVGA.DRV) supporting the high-resolution and 256 color modes of most popular VGA chipsets. The Super VGA mini-driver should be installed when Setup detects VGA. However, it is not a good idea to automatically run in a Super VGA mode, since many customers do not need or want 256 colors, which runs significantly slower than 16 colors. Also, the installed monitor may not support resolutions higher than 640x480.

An easy user interface should allow the user to select a Super VGA display mode. Some amount of display hardware detection can be applied to not allow a mode that the current adapter does not support. However, since the monitor cannot be detected, it is still possible that the user will pick a mode their adapter supports, but not be able to see on their monitor. Display selection must be "foolproof" to the extent that if a customer selects a mode that their adapter or monitor does not support, Chicago should fall back to the original display mode.

The Super VGA mini-driver should provide the following display modes:

640x480, 256 colors (512K)
800x600, 16 colors (512K)
800x600, 256 colors (1Mb)
1024x768, 16 colors (512K)
1024x768, 256 colors (1Mb)

Support should be built in for the following Super VGA chipsets/adapters:

Tseng ET4000 chipset adap	Diamond Speedstar Orchid ProDesigner II

Western Digital / Paradise VGA (90C11, 90C20, 90C30 chipsets)	Paradise VGA Professional Paradise VGA 1024 Misc. OEM

HIGHLY CONFIDENTIAL

MX 2120560
CONFIDENTIAL

ATI VGA	VGA Wonder series Misc. OEM
Video Seven VGA	FastWrite 1024i VRAM VRAM II

HIGHLY CONFIDENTIAL

MX 2120561
CONFIDENTIAL

CMS 00013855

RBC 04256

Trident 8900 A	Trident Impact series Misc. OEM
Oak Technology 077	Misc. OEM

Status

A generic 256 color display driver (SVGA256.DRV) is being developed for Windows 3.1a and the Windows Driver Library, and scheduled to be available by 1/93. The driver is nearly complete for the chipsets listed above, but a VDD is still under development. SVGA256.DRV provides the bulk of the device-specific initialization and bank switching code for the SVGA.DRV mini-driver.

Open Issues

HIGHLY CONFIDENTIAL

MX 2120562
CONFIDENTIAL

CMS 00013856

* What other chipsets should be added? Biggies include:

Tseng ET3000 (Orchid Prodesigner)
Cirrus Logic (notebook PCs)

10.2.5. Fall Back Display Mode

Priority 1

Objective

If a display driver fails to load under Windows 3.1, the user is kicked back to MS-DOS. But Chicago doesn't have an MS-DOS to get kicked back to! If for any reason an incorrect display driver is installed, Windows should fall back to VGA or another appropriate standard display driver. This feature will help PSS get customers to the Windows desktop easily and quickly, where the real display configuration problem can be identified and resolved.

Feature Description

Setup should detect the maximum standard and "in the box" display driver usable on the current display. In most cases, this will be EGA or VGA, which are built into the Universal Display Driver (DISPLAY.DLL). Whenever the actual display driver fails to load, Windows should load the fall-back driver instead, and an immediate error message dialog should appear before the shell loads. This should tell the user that the specified display mode could not be set, and that they should use the Control Panel Display icon to reconfigure their display options. The user should be able to jump to the Display dialog to reconfigure their display immediately.

Fall-back display mode should also be accessible by holding down a hot-key during startup. This will allow users to bypass display problems that will not cause the display driver to fail to load. For example, this can happen if the attached monitor does not correctly synch to the specified screen resolution.

Status

Development must design this into GDI and kernel display driver loading code.

Open Issues

HIGHLY CONFIDENTIAL

MX 2120563 CONFIDENTIAL

10.2.6. GDI DIB Engine Drivers

Priority 1

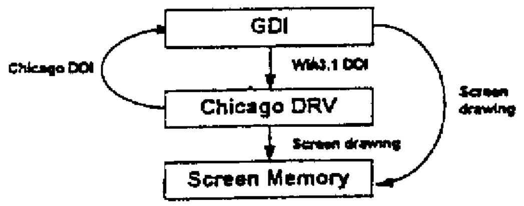
CMS 00013857

RBC 04258

Objective
Smaller, faster, more robust display drivers for DIB frame buffer display devices, especially 24 bit displays.

Feature Description
Chicago's GDI will provide a set of optimized, generic drawing functions for 8 bit per pixel DIBs and higher. Used in conjunction with minor display DDI changes, these services will allow fast development of frame buffer display drivers. The Universal Display Driver will use the GDI DIB Engine. IHVs who choose to not use the Universal Display Driver for frame buffer displays will still benefit by using the DIB Engine.

A Chicago display driver will be called by GDI using an extended version of the existing Windows 3.1 DDI. As in Windows 3.1, the driver can tell GDI which capabilities it supports internally, and which should be simulated by GDI. However, the DIB engine services provided by Chicago GDI will be callable by the display driver in order to handle common screen drawing operations:



Status
Development is well underway by AmitC and RayPat. The DIB engine is scheduled for completion in 1/93, with several mini-drivers build along the way.

Open Issues

10.2.7. Display Control Panel

Priority 1
Objective
Installation and configuration of display adapters and drivers is very difficult in Windows 3.1. Drivers are linked to chipsets and brand names, and installed from maintenance mode Setup in MS-DOS and Windows, plus IHV supplied programs.

Chicago must provide "one stop shopping" for all display related installation and configuration. Users should be isolated from display chipset, adapter, and brand names.

Windows NT (Release 2) and Chicago should use the same display UI.

Feature Description
Display driver installation in MS-DOS and Windows maintenance mode Setup should be eliminated. A single Display icon in the Control Panel should provide all display-related configuration features. This includes:

* Adapter installation & configuration

* Display mode selection: color, resolution, 3rd party features
* Scaling features (if supported)
* Extra device-specific IHV options dialogs
* Monitor configuration (if monitor drivers are used)
* Driver updates
* Driver information: IHV, version, etc.

Users should see a list of display modes that they can set, independent of adapter type. This is dependent on adapter and monitor capabilities.

The Configuration Manager should track the capabilities of the adapter and monitor. Then, the Display Control Panel will rely solely on the Configuration Manager to supply a list of display modes for the user to select from.

Status
Design proposal under development. The Display team will cooperate with the Shell/UI team for this.

Open Issues

10.2.8. Easier Driver Installation
Priority 1
Objectives
Simpler display driver installation for displays not supported in the Chicago box.

Reduced COGs for OEM driver disks by eliminating fonts, logos, etc.

Feature Description
Display driver installation should be built into the display Control Panel, and provide the following features:

* <u>Windows 3.1 OEMSETUP.INF Compatibility: The utility should be able to install display drivers from Windows 3.x version OEMSETUP disks.</u>

* <u>No more FON files: Display drivers will not determine screen DPI in Chicago, as</u>

<u>they do in Win3.1. Rather, this will be determined by GDI and a possible CPL UI. For this reason, system fonts are no longer linked to the display driver. However, not including fonts breaks compatibility with Win3.x.</u>

<u>* No more LGO or RLE files: The startup logo should not be linked to the display driver. Logo code for standard EGA and VGA</u>

display modes will suffice for the Chicago customer base. WIN.COM and/or Setup should auto-detect the correct display mode for the startup logo. Not including logo files breaks compatibility with Win3.x.

* No more 2GR files: Standard mode grabbers are not used in Chicago. Not including 2GR files breaks compatibility with Win3.x.

This should be investigated and developed by Plug and Play team as part of the Configuration Manager and Display Control Panel.

Open Issues

10.2.9. Pen Compatibility

Priority 1

Objective

All Chicago display drivers will support the small set of extensions required to support Windows for Pen Computing. The Mobile Windows group should not have to maintain a separate set of display drivers for Chicago.

Feature Description

Inking requirements for display drivers are documented in the Windows 3.1 DDK, and may be extended for Chicago.

Status

Code is checked into Windows 3.1 drivers, but not enabled. This should happen early on in Chicago for compatibility testing purposes.

Open Issues

* Mobile Windows group should be responsible for adding new pen capabilities to Chicago display drivers.

10.2.10. Device Independent Color Support

Priority 1

Objective

Chicago should supply a device driver interface to allow display drivers to match/correct colors to the current printer. However, "in the box" drivers will not necessarily support this feature.

Feature Description

TBD by DavidW and the GDI team.

Status

DavidW has investigated and reported on this. EricBi will program manage device independent color. KeithLa will oversee integration of this feature into Chicago display drivers.

Open Issues
* Which in-the-box drivers should support DIC?

10.2.11. Universal VDD

Priority 1
Objective
In addition to lots of great display drivers, Chicago needs to have broad VDD support. A great source of customer dissatisfaction with Windows' MS-DOS application support stems from inadequate built-in VDD code for Super VGA chipsets. Our previous policy of leaving this up to the IHVs is not good enough.

Feature Description
Chicago should provide a "Universal VDD" (VDD.386) analagous to the Universal Display Driver, as follows:

* Universal VDD: VDD.386 (or simply *VDD) should support standard EGA and VGA displays. VDD.386 should be installed on all EGA or VGA compatible Chicago PCs.

MX 2120570
CONFIDENTIAL

* Mini-VDDs: An interface should be provided to load multiple "mini VDDs".

CMS 00013864

HIGHLY
CONFIDENTIAL

RBC 04265

These will provide save/restore support for additional display hardware and extended modes. Mini-VDDs will be specified in SYSTEM.INI. Mini-VDDs should be included to support all other display adapters supported in Chicago, e.g. XGA, S3, etc. Unfortunately, vendors are terrible at this, and hate to do it. We should write as many mini-VDDs as

<u>possible internally, with the cooperation of IHVs.</u>

<u>* Super VGA Mini-VDD: VDDSVGA.386 should be a mini-VDD providing standard VGA display mode save/restore for all adapters supported by the SVGA.DRV display mini-driver. VDDSVGA.386 should be installed whenever SVGA.DRV is installed.</u>

* Standard VGA Display Modes: Chicago VDDs must save and restore all standard VGA display modes. Standard text modes must run in a window.

* Non-Standard Display Modes: Chicago VDDs must allow non-standard display mode VMs to run. However, if no VDD is loaded which can restore the extended mode VM, then the user

should be warned before the VM starts that they will have to exit the VM in order to return to Windows. This will prevent possible crashes and hangs. Not many customers run non-standard display modes in VMs; however this is popular with MS-DOS graphics programs, such as AutoCad.

* Windowed VGA Graphics Modes: Many Super VGAs and display accelerators

CMS 00013868

<u>cannot support this feature. We should not spend cycles on this due to inherent h/w and performance limitations. It has not emerged as a compelling end-user or competitive feature. We need to shift this burden onto IHVs.</u>

Status

A company called Engenious Solutions is being contracted to build VDDSVGA.386, a VDD supporting many Super VGA chipsets. This will be used in conjunction with the generic 256 color driver, SVGA256.DRV. VDDSVGA.386 will provide the base for the Chicago Super VGA mini-VDD.

Open Issues

<u>* How much effort should we apply to supporting extended</u>

HIGHLY CONFIDENTIAL

MX 2120575 CONFIDENTIAL

CMS 00013869

RBC 04270

mode VMs for Super VGAs, etc.?

10.2.12. Standard Grabbers

Priority 1

Objective

Generic, full-featured, high-performance grabbers in Chicago should be used by all vendors, eliminating the need for third party versions.

Feature Description

Chicago should eliminate the duplicate grabber code currently existing in Windows. All Chicago VDDs should be updated to at least the Windows 3.1 model (if not the Chicago model), and use a single grabber. This will probably be VGADIB.3GR. Windows 3.1 contains several widely used enhanced mode grabbers:

V7VGA.3GR Win3.0 VDD model; used by V7VGA, XGA, and many WDL drivers. Does not support graphics mode MS-DOS applications in a window or smooth scrolling. This can be eliminated by updating all Chicago VDDs to the Win3.1 model or later.

VGA.3GR Win3.1 VDD model; used by VGA only. Supports graphics mode MS-DOS applications in a window and smooth scrolling.

VGADIB.3GR "DIB" grabber; Windows 3.1 VDD model; Used by 8514, EGA, and TIGA. Supports graphics mode MS-DOS applications in a window and smooth scrolling.

VGA30.3GR Windows 3.0 VDD model; used with VDDVGA30.386 by "VGA (Version 3.0)" and "XGA (640x480, 16 colors". This was provided for backwards compatibility with chipsets not directly supported by Win3.1 VDD, but broken by it. This can be eliminated by supporting most chipsets directly in the Chicago VDDs.

Chicago grabbers should support the following features:

- Windowed Mouse Support: All Chicago grabbers will support the mouse for windowed MS-DOS applications.
- Smooth Scrolling: All Chicago grabbers will support the smooth scrolling feature added to VGA.3GR and VGADIB.3GR for Windows 3.1.
- Text Sizing All Chicago grabbers will support text sizing for windowed MS-DOS applications.

10.2.13. Dynamic Screen Dimensions

Priority 2

MX 2120576
CONFIDENTIAL

HIGHLY CONFIDENTIAL

CMS 00013870

RBC 04271

Objective
Users should be able to switch display dimensions (e.g. 640x480, 800x600, 1024x768, etc.) without restarting Windows. This is already being partially acheived in Windows 3.1 by the Radius Pivot product.

Even with the Physical/Scalable WYSIWYG goals described above, users are still likely to need/want to change screen resolution. For example, when switching from LCD to external CPU on a notebook computer. Also, this would benefit pen based PCs.

Feature Description
Affected components to support on-the-fly screen resolution changes:

* <u>Control Panel: A new, generic, centralized user interface should allow users to change to another supported screen resolution and color depth.</u>

* <u>GDI: Core support is needed.</u>

* <u>Display Drivers: GDI should notify the display driver</u>

MX 2120577
CONFIDENTIAL

HIGHLY CONFIDENTIAL

CMS 00013871

RBC 04272

through a new DDI that the user has requested a different resolution.

* Applications: Applications may need to resize their windows, etc.

Status
Open Issues

10.2.14. Dynamic Color Depth
Priority 2
Objective
Chicago should allow users to change their screen color depth without restarting Windows. Many customers have requested that pixel depth be changeable "on the fly" so that they do not have to exit their applications, save their documents, restart Windows, and then reopen their applications and documents. The Macintosh has supported this feature since the Mac II added color support to the originally black & white Macintosh line.

A popular application of this feature will be switching to an external CRT from an LCD screen on-board a notebook computer. Desktop publishers tend to use high-color displays so that their color art and photographs can be displayed in their on-screen layouts. However, the screen redraw time for this is very slow in high color. They seek to do their layout work in monochrome, and quickly switch to full color for preview. Also, this has applications in multimedia, in which a different color depth might be required to display a certain type of picture or animation.

Feature Description
Under Windows 3.1, to change the color depth of the screen (e.g. from monochrome (1 bpp) to 256 colors (8bpp), the display driver must be changed (or one of it's input parameters modified) and Windows must be restarted for the changes to take effect. It used to be that separate display drivers were required for different pixel depth display modes. This required Windows to be restarted so that the core modules (USER, GDI) could re-link to the different driver module. However, non-VGA display technologies and improved driver code are allowing single display drivers to support multiple color depths. The Windows 3.1 8514, XGA, TIGA, ATI/Ultra, DGIS, RGDI, and S3 display drivers work this way

Affected components to support on-the-fly color depth changes:

* Control Panel: A new, generic, centralized user interface should allow users to change to another supported screen resolution and color depth.

* GDI: Core support is needed.

* Display Drivers: GDI should notify the display driver through a new DDI that the user has requested a different color depth.

* Applications: Applications store off-screen bitmaps in device dependent (DDB) format for performance reasons. These applications must be notified when the format of the DDB changes color depth.

Status

Open Issues

* Is this really an important feature? Do most users really need to switch frequently?

MX 2120580
CONFIDENTIAL

CMS 00013874

HIGHLY CONFIDENTIAL

RBC 04275

Greatest usage is LCD => CRT changes.

10.2.15. Physical / Scalable WYSIWYG

Priority 3

Objective

Customers complain about two things in particular relative to the appearance of Windows on displays:

1) Text and objects are too big or too small.

2) Text and objects on-screen do not exactly match their printed sizes.

Two feature can be added to Chicago to allow the user to fine-tune their display to their own visual preferences:

Physical WYSIWYG: The user should be able to have screen text and other objects exactly match their printed sizes. By having the user precisely measure and/or report the actual density of pixels on their monitor, it will be possible for GDI and applications to provide "physical WYSIWYG", i.e. text and graphics size on screen should exactly match that of a printed page.

Scalable WYSIWYG: Screen resolution (e.g. 640x480, 800x600, 1024x768, etc.) should become meaningless and transparent to the user. That is, users should not have to select a resolution in order to meet their preferences for readability and desktop "real estate" (how much they can see on screen). Instead, they should simply run at the highest possible resolution supported for their color needs, adapter, and monitor. This provides maximum desktop real estate, and other objects on screen should be scalable in order to provide greater readability, avoid eyestrain, etc.

Feature Description

Under Windows 3.1, display drivers provide logical and physical values for the "dots per inch" resolution of the display device. The physical value is used by GDI in all mapping modes other than MM_TEXT to determine the pixel sizes of on-screen objects. The logical value is used by applications to scale their on-screen text and other graphics objects. For example, the VGA driver uses a 72 dpi physical resolution and a 96 dpi logical resolution. For this reason, a 1 inch line drawn by applications in MM_TEXT mode are larger in pixel size than those drawn by GDI in other mapping modes. This provides for improved readability by enlarging text on-screen. However, this means that screen text is larger than printed text.

Unfortunately, there are several problems with having the display driver provide physical and logical screen resolutions:

* <u>The driver does not know the physical size of the monitor, so cannot accurately specify the number of actual pixels per millimeter.</u>

* <u>The user has no choice in the logical scaling of their graphics, and thus are stuck with 96 dpi.</u>

Chicago should remove the specification of screen resolution from the display driver, and replace it with a Control Panel user interface to achieve "physical and logical WYSIWYG" according user preference and specification. Users can opt to have Windows scale their text and screen drawings to their own preferences, while retaining "logical WYSIWYG", i.e. text and graphics sizes will be relatively proportionate on-screen as on printed page For example, the WinWord ruler will show correct measurements, even if it is not physically accurate.

Affected product areas include:

MX 2120582
CONFIDENTIAL

HIGHLY CONFIDENTIAL

CMS 00013876

RBC 04277

*   <u>GDI: Core support required to initialize correctly, scale drawing, etc.</u>

*   <u>Display Drivers: Drivers can get out of the screen DPI business That is, GDI ignores the resolution reported by the device (or uses it as a default). All that display drivers are concerned with are how many dots are on the screen, not how big they are.</u>

* Control Panel: The Control Panel should provide a user interface to allow the user to easily and intuitively specify the scaling factor GDI should apply to the logical DPI of the screen device. Setting physical DPI = logical DPI will result in "physical WYSIWYG" provided that GDI has been correctly calibrated to the physical resolution of the screen.

* Screen Resolution API: Applications will not have to support a new API for physical and logical WYSIWYG. GDI will report these resolution values through existing API.

* SYSTEM.INI Section: [Display]: Chicago will standardize the specification of device resolution information using profile values in the [Display] section of SYSTEM.INI:

PhysicalDPI=...

The Control Panel will set this to a value as derived from the user's measurement of their physical screen resolution. GDI will read this value at start-up, and use it instead of the values reported by the display driver. If this line is not present, the display driver value will be used by default.

LogicalDPI=...

The Control Panel will set this to a value as derived from the user's scaling preference for their logical screen resolution. GDI will read this value at start-up, and use it instead of the values reported by the display driver. If this line is not present, the display driver value will be used by default.

Status
DavidW has submitted a memo on this subject.

Open Issues

* Many compatibility issues?

10.2.16. Monitor Drivers

Priority 3

Objective

Display capabilities are a function of the installed monitor as well as adapter and drivers. Chicago needs a way to integrate monitor capabilities into its display features.

Feature Description

Monitor drivers should be text file descriptions of monitor capabilities, including size, color and frequency information. NT will use the VESA "Video Display Device Protocol" (VDDP) spec. VDDP files have filename extensions of ".VDP". VDPs should be considered for Chicago.

Monitor drivers could benefit the following Chicago features:

- Display Initialization: Display drivers can read the monitor file to determine the optimal frequency for the selected display mode. For example, VESA "ergonomic" modes can be set automatically. At present, display adapters ship separate MS-DOS utilities, such as VMODE.EXE, to allow the user to select the display frequency.

- Display Mode Selection: The Display Control Panel and/or Configuration Manager can use monitor info to allow selection of only the display modes supported by the attached monitor; or, messages could appear warning the user that their monitor might not support the selected display mode.

- Device Independent Color (DIC) support may make use of monitor drivers.

Multiple monitor drivers should be installable. SYSTEM.INI will designate the current monitor in use. If Chicago allows multiple simultaneous displays, this DDI will be extended.

A set of default monitor drivers should be provided for generic support, e.g.:

Default 14", Color
Default 15", Color
Default 16", Color
Default 17", Color
Default 19", Color
Default LCD, 16 Grayscales
Default LCD, 32 Grayscales
Default LCD, Color

Monitor drivers should be changeable by text editor only.

MX 2120588
CONFIDENTIAL

Status

This should be investigated by the Plug and Play development team as part of the Configuration Manager

Open Issues

10.3. Driver Summary

Under Chicago, display adapters will be supported by three types of drivers:

• Windows 3.1 Display Driver: This may limit performance on some new Chicago graphics features. See feature description "Windows 3.1 Compatibility".

• Display Mini-Driver: The easiest way to provide support for flat or bank-switched Super VGA frame buffer displays is to take advantage of the new Universal Display Driver. A mini-driver provides initialization, bank-switching, and BitBLT acceleration. See feature description, "Universal Display Driver".

• Chicago Display Driver: Compatible only with Chicago. Essentially the same DDI as Windows 3.1, but may use the GDI DIB engine and support additional driver interfaces, such as device independent color. See feature description, "GDI DIB Engine".

In addition to one of the above modules, displays must have proper VDD support. See the section, "Universal VDD".

All Chicago display and mini drivers will be coded to run only on 386 compatible CPUs.

Standard mode grabbers (*.2GR) are no longer required and will be removed.

10.3.1. Displays No Longer Supported

The following display types will not be supported in the Chicago box, since they are not common on 386 based PCs. Compatible drivers will be made available via the WDL according to demand.

* <u>EGA black and white (286 only)</u>
* <u>EGA Monochrome (286 only)</u>
* <u>Hercules Monochrome</u>
* <u>IBM MCGA (286 only)</u>

* Olivetti/AT&T Monochrome or PVC Display

* QuadVGA, ATI VIP VGA, 82C441 VGAs

10.3.2. Broad Super VGA Support

A Super VGA mini-driver will support at least 90% of Chicago customers with Super VGA display adapters, i.e. 256 colors or 800x600/1024x768 resolutions. See the feature description, "Super VGA Mini-Driver" for a listing of Super VGA displays supported generically in Chicago.

The proportion of Windows 3.1 users with displays capable of resolutions and colors beyond standard VGA (640x480, 16 colors) is estimated to be 40%, or about 4 million PCs (this is probably a low estimate). This includes Super VGAs, plus "fixed function" accelerators such as 8514, XGA, and S3. The breakdown of these displays is shown below. (These numbers are highly speculative estimates based on information provided by several top vendors to Windows Program Management and the Developer Relations Group.)

Type	High Estimate	Low Estimate
Tseng ET4000 VGA	25%	20%
Western Digital / Paradise VGA	20%	15%
S3	15%	10%
ATI Ultra	15%	10%
ATI VGA	12%	8%
Video Seven VGA	10%	5%
Trident VGA	7%	3%
Oak VGA	5%	2%
8514/A (IBM, Paradise)	5%	2%
XGA (IBM)	5%	2%
Other: Low-end VGAs, TIGA, DGIS, etc.	5%	1%

The bold-faced adapter types above are Super VGA displays, comprising approximately 65% of high resolution displays. Display accelerators (S3, ATI Ultra, 8514/A, XGA) total approximately 30%.

By Chicago release, these numbers are estimated to change as follows:

- 75% of Chicago capable PCs (386/4Mb) support some type of 256 color and/or high-resolution display mode.

* <u>Super VGA adapters will constitute at least 60% of high-resolution displays, the other 40% will be accelerators. This is due to extensive usage of Super VGAs by OEMs in their base model PCs.</u>

* <u>S3 share will stay steady at 10 - 15%.</u>

* <u>ATI Ultra will lose share due to competing accelerator products, down to 5-10%.</u>

* <u>Tseng's new ET4000/W32 accelerator will probably be very popular, at least 5%.</u>

* <u>XGA will gain share, up to around 10% of high-resolution displays.</u>

10.3.3. Flat Frame Buffer Displays

Display chipsets which provide flat DIB frame buffers will be supported with mini-drivers which utilitize the GDI DIB engine for drawing. Mini-drivers for the following flat model chipsets are likely to be included in Chicago:

* <u>Chips & Technologies "Wingine"</u>

MX 2120592
CONFIDENTIAL

HIGHLY CONFIDENTIAL

CMS 00013886

RBC 04287

* Dell DGX ("Jaws")

10.3.4. S3 Adapters
These are popular retail upgrade products, and an increasing number of OEMs are offering them standard in high-end models. Drivers for S3 adapters will be included in Chicago. This GUI accelerator chipset is VGA compatible, but its Windows graphics modes are more akin to 8514 or XGA. This driver will probably not be a mini-driver. S3 will determine the most appropriate way to support their adapters in Chicago.

10.3.5. 8514/A
This Windows 3.1 driver will be updated to support Chicago features well and use the DIB engine. IBM announced in 8/91 that they are dropping support for 8514/A, but still ship it in some PS/2 models.

10.3.6. XGA
Chicago's XGA driver be expanded to support all IBM XGA, XGA/2, and VESA XGA specifications. All standard XGA modes will be supported. XGA may be very important by Chicago release, as Intel has licensed the design from IBM for incorporation into future chips.

10.3.7. ATI Ultra, Vantage, 8514 Ultra
ATI's drivers for these adapters will be included in Chicago. These will not be mini-drivers.

10.3.8. Video Seven Super VGA
V7 adapters will be supported by the Super VGA mini-driver instead of the V7-specific drivers contained in Windows 3.1. Chicago's will be faster.

10.3.9. Compaq Portable Plasma
These displays are on Compaq "lunchbox" PCs, we need to evaluate if these will be used with Chicago.

10.3.10. VGA (Version 3.0)
This was included in Windows 3.1 for compatibility, it is the 3.1 DRV with 3.0 VDD and 3GR. The need for this should be eliminated in Chicago.

10.3.11. Displays Under Consideration
The following miscellaneous third party display types will be considered for Chicago, based on market share, disk space, etc.

* Tseng ET4000/W32: The new Tseng Windows accelerator.
* Paradise Windows Accelerator (WD90C31)
* Weitek Windows Accelerator

10.3.12. Displays Unlikely to Be Supported "In the Box"

By the Chicago timeframe, the following display types probably will not have compelling market share to justify drivers in Chicago. However, these adapters should be supported via the WDL.

- TIGA (Texas Instruments)
- DGIS (Graphic Software Systems)
- Appian RGDI

10.4. DDK Plan

Windows 3.1 customers suffered from the lack of compatible drivers which could take advantage of new features in the months immediately following final release. To avoid this problem for Chicago, we should strive to achieve several goals, described below.

10.4.1. Comprehensive, High-Quality Documentation & Source Code

All important new and modified DDI must be demonstrated in the first pre-release of the Chicago DDK. This means that the design must be almost final, with only bug fixes and minor feature additions and DDI changes included in follow-up releases.

The Windows 3.1 DDK provided some sample source and docs that were obsolete or incomplete. For example, MOUSE.DRV in the DDK was not the same used for the retail version due to propietary features sold by SPAG. Also, changes to MOUSE.COM (the MS-DOS level mouse driver) to support the mouse for windowed MS-DOS applications was not fully documented in the DDK due to late spec addition. The Chicago DDK should be sure to include demonstrations of all basic DDI and features.

For displays, the following source code should be included in the DDK.

MX 2120594
CONFIDENTIAL

* Flat Frame Buffer Mini-Driver: This is likely to be for the C&T Wingine, in order to demonstrate application of the Universal Display Driver to simple flat frame buffer displays.
* Super VGA Mini-Driver: IHVs will be able to add h/w specific performance optimizations to our high-speed, full featured generic driver.

* XGA: The XGA standard is likely to be very widespread by Chicago release. Developer Relations reports that most major display chipset vendors are planning XGA compatible products. Also, Intel will integrate XGA into future CPU chipsets.

* Sample Mini-VDD: IHVs traditionally have great difficulty trying to modify our VDDs to support their

<u>chipsets, and usually do it poorly. A single, well-documented mini-VDD should be included that demonstrates how to plug additional chipset support into Chicago. Note that this should not be the Super VGA mini-VDD, IHVs should write their own mini-VDD, rather than add support to our generic version.</u>

<u>* Generic Grabber: We should discourage IHVs from</u>

<u>producing their own grabbers. However, some may seek to support proprietary video modes for windowed MS-DOS applications. A single, well-documented grabber should be included in the DDK.</u>

10.4.2. Early Delivery

The DDK is as important as the SDK in terms of third party support for new Windows features and performance increases. Windows 3.1 taught us that IHVs are as tight with resources as we are, and reluctant to assign engineers to driver revisions from pre-release code when the next product is on the line. Since many of the "in the box" drivers are likely to come from IHVs, it is critical that near-final DDK docs and sample source be shipped in the Beta 1 time-frame, at least 6 months before final release.

DDK delivery should be made in the following stages:

* <u>Partner Alpha: At least 9 months prior to release: IHVs contracted to deliver "in the box" drivers should</u>

receive this undocumented source code release. Includes only soft-copy draft docs (*.TXT, *.WRI, *.DOC, etc.)

* Beta 1: At least 6 months prior to release: General release to all beta IHVs. DDI complete, but possible code changes. Includes soft-copy DDAG (*.HLP).

* Beta 2: At least 2 months prior to release: General

release to all beta IHVs. DDI and code complete, bug fixes only after release. Includes final soft-copy DDAG (*.HLP).

10.4.3. Developer Support and Updates

The Windows 3.1 SDK beta received excellent support from PSS/DS via OnLine and CIS. For Chicago, the DDK should receive a similar level of support, including prompt delivery of source code updates.

* Developer Support via OnLine for Alpha Partners

* Developer Support via CIS and OnLine for Beta IHVs

* Source Code Updates via CIS and OnLine

MX 2120600
CONFIDENTIAL

HIGHLY CONFIDENTIAL

CMS 00013894

RBC 04295

11. Printing Changes

11.1. Executive Summary

This is the second draft of the specification for all printing changes proposed for Chicago. It is currently intended to serve as an outline with much detail still to be provided. Time estimates are provided for those issues requiring additional detail.

The goals are to address requests made from our customers: end users, ISVs & IHVs. These requests fall into 3 primary areas: Improved performance, great network printing, and ease of use. Printing is an area that had not been given much detailed attention until Windows 3.1, and we need to expand upon the improvements made in Windows 3.1. The main goals that all of the proposed changes are attempting to address are:

- Simplify printer driver installation/configuration/use (Plug-n-Play)
- Improve printing performance by decreasing time needed to return control to application.
- Address customer issues raised against Windows 3.1
- Accommodate WPG's RBPA system requirements
- Migrate non-UNIDRV based printer drivers to UNIDRV
- Add support for printers shipped post-3.1 based on sales data
- With all of the above, ensure solutions are consistent with NT (i.e. no divergant solutions)

Features

The changes noted above can generally be placed into 3 areas, GDI/System level changes, Printer Driver changes, and miscellaneous printing issues. As noted above, there are several areas requiring additional information from development that are noted where applicable. A brief description of proposed major features is described below, detailed discussions may be found under the noted section in this document. The number following each description indicates the priority associated with it: 1=Must do, 2=Should Do, 3=Do if time available.

As a reference, see GDI Printing Architecture, also in the Chicago Specification.

GDI/System changes
- System support for plug-n-play installation/configuration of printers (1)
- System support for Bi-Directional communication with Printer (1)
- Spooling Metafiles: Decrease time for return of control to application when printing. (1)
- New print subsystem. (1)
- Multitasking Print manager with improved UI. (1)
- Device Independent Color: Allows screen-to-printer color matching (1)

Printer Driver changes
- Improved UI for UNIDRV & Postscript drivers
- Support all important non-Postscript printers with UNIDRV
  - HP LaserJet 4 (Payette) UNIDRV support & minidriver (1)
  - Canon CaPSL UNIDRV support & minidriver(2)
  - PPDS (Lexmark printer command language) UNIDRV support & minidriver (2)
  - UNIDRV support to replace current TTY.DRV with minidriver (2)
- UNIDRV support for Device Independent Color system changes (1)
- UNIDRV support for Bi-Directional communication with printers (1)
- Limited Postscript level 2 support (Color support & data compression) (1)
- Allow Postscript driver to read PPD files in addition to WPD files (1)
- Postscript driver:Decrease code size & increase speed
- New mindrivers as appropriate (3)
- Replace HP Plotter Driver (3)

Miscellaneous Changes
- Windows Printer Group issues (2)
- PSS issues (2)

11.2. "Plug & Play"

11.2.1. Objective

Installing new/updated printer support is currently not an easy process for users under Windows 3.1. We need to make this as simple and painless as possible in *all* cases. There are 3 interesting scenarios for Plug-n-Play for printer support. The first is for locally connected output devices that support Bi-Directional communication with the host, and the second is for locally connected devices those that don't support BiDi, and the last is for printers available via the network.

There are currently 3 classes of printers that support "BiDi" communications: Postscript printers connected via a serial port (connecting to a parallel port loses this ability). Printers that provide their own PDL (Page Description language) & are bidirectionally aware (like the HP LaserJet 4, aka Payette), and RBPA (Resource Based Printing Technology) type printers like Jumbo & Dumbo from WPG. Also note that each of these may be connnected either directly or across the network.

For all other locally connected printers, which are our installed base today, Plug & Play solutions will require some user intervention, which we should keep to a minimum. However, there are some very basic questions novice users typically have that we can & should address under Chicago. These questions are: What port is my printer connected to? What communications settings should I use (for serially connected printers)? What printer driver should I use (for the completely clueless)? What printer driver should I use (for those able to determine that what they have is not on the list of supported printers)? This last question is one of the single most asked questions of PSS, and one they are equiped to answer the least. Providing ways for non technical users can answer these questions for themselves would save hundreds, if not thousands, of calls to PSS over time.

For printers not connected locally, we need to provide a mechanism for providing a "friendly" printer name that network administrators may define. A user at a workstation should then be able to browse thru printers available within their domain (via the "friendly" names or machine names for Technoids), select a particular printer, and have the printer driver automatically get copied from the server to the workstation along with any needed configuration data. The network administrator would use one of the 1st two methods to determine the correct driver to use when installing on the server.

11.2.2. Proposed Solution

11.2.2.1. Overview

Several details stil not worked out at this point, most importantly how this will work across a network, but generally are as follows:

1) System queries all local ports to see if any "Bi-directionally aware" devices are installed.
2) If a Bi-directionally aware device is found, system performs protocol negotiation & establishes faster possible Bi-Directional protocol available to both host & device.
3) Once protocol established, get Device ID value & examine registry to determine appropriate device driver(s) & additional components required, and install them into system without user intervention, unless a floppy is required & then need to prompt user for it. Installation of driver is complete at this point & driver should auto-configure itself.
4) If a Bi-Directionally aware device is not found, prompt user with minimal amount of questions to determine which driver to use. This would start with "Do you know the name of the printer that you want to use?", which depending upon the level of expertise the user has, may digress to to the point where we simply try an Epson 9 pin driver, an Epson 24 pin driver, etc. This process will involve determining which port a printer is connected to. A surprisingly large number of calls are recieved by PSS where customers can't figure this out. We can add a button called "Test Port" or something like that to the driver install dialog that prints "This is LPTx:" and/or addtional diagnostic data, so users can figure this out w/o calling PSS.

11.2.2.2. Affected Components

Setup &/or Control panel (whatever component winds up being responsible for installing new printer drivers). Also need COMM protocol handlers for all Bi-Directional protocols supported (currently should plan on Zippy, Peppy, Boise Nibble, Boise Byte & Jumbo). Need some form of device registry that associates device ID values reported from BiDirectionally aware devices with both device drivers in the retail product (like the HP LJ4) and with devices we don't provide drivers for (like Jumbo & Dumbo). UNIDRV.DLL & PSCRIPT.DRV may need to have functionality added to allow them to process data recieved back from the printer (or this may be handled elsewhere & communicated to the driver via an ExtDeviceMode() call). Effect on network components unclear at this time.

11.2.3. Outstanding Issues

Need to determine the following:
1) Which component is responsible for protocol negotiation? Will this be extensible for protocols developed after Chicago ships?
2) Who will provide the protocol handlers for COMM.DRV?
3) Which component checks for device ID's & where does it get the information to compare what is returned by the device against?
4) How do we handle an unknown device ID?

5) Need a detailed algorithm we should use for non-BiDi printers
6) Current WPG Bi-Directional protocols are all based on either standard parallel ports (Jumbo, Boise & Peppy), or enhanced parallel ports (Zippy) on host, need to determine if appropriate to extend this to serial ports. Network connection implications unclear.
7) Need details for network printer support.

11.2.4. Current Status

11.2.4.1. Overview

While the general approach to Plug & Play support for printers has been outlined, significant amounts of detail remain to be filled in. EricBi owns working with JohnPa & coordinating with WPG to provide a detailed plan.

11.2.4.2. Schedule

Detailed plan complete by 10/31/92. Additional schedule TBD based on that plan.

11.3. Bi-Directional Communication with Printer

11.3.1. Objective

To allow device driver support of new Bi-Directional communications protocols on parallel ports being developed by Microsoft & Hewlett-Packard. This support will allow support of the RBPA (Resource Based Printing Architecture) being developed by WPG (Windows Printing Group), allow better error reporting, allow the driver to automatically configure itself based on info received from the printer, and enable resource based printing. RBPA is a technology developed by WPG requiring Bidirectional communication with the printer that allows the host & printer to intelligently share processing required to generate output. A more detailed discussion is beyond the scope of this document, additional information available from WPG.

11.3.2. Proposed Solution

The proposed COMM changes for Chicago will allow for easy installation/addition of specialized communication protocol handlers in Chicago. See \\PYREX\PROGMAN\SRC\COMM\CHICOMM.DOC for a complete description). The new print subsystem described elsewhere in this document takes other system architecture concerns from WPG into account. The sum of these changes will provide the necessary system level support for BiDirectional communications with a printer. Since the RBPA work being done by WPG will provide their own specialized device drivers, the only remaining issue is what functionality is appropriate to be added to printer drivers in the Chicago product (i.e. UNIDRV.DLL & PSCRIPT.DRV) to support this in non-RBPA type printers (like the HP LJ4 & Postscript devices).

11.3.3. Outstanding Issues

1) How to support Bi-Directional communications with printer under the Universal Printer driver architecture.
2) How to support Bi-Directional communications with printer with the Postscript driver.

11.3.4. Current Status

11.3.4.1. Overview

Currently waiting to examine the HP LJ4 driver to examine their approach to this. We may be able to accomplish this without explictly changing the printer drivers, and have another component query the device for configuration information & then update the driver via an ExtDeviceMode() call.

11.3.4.2. Schedule

Waiting to examine the HP LJ4 bidirectional driver to examine their approach to this. Need to research appropriateness of having Postscript driver support bidirectional communication.
EricBi owns researching this & following up with development, proposal expected by 1/1/93 (developement won't require this info untill then)

470	Bi-Directional printing - Driver Changes	1	EricBi

11.4. Spooling Metafiles

11.4.1. Objective

Users currently must wait an indeterminate amount of time after issuing a "print" command from an application before control is returned to them. The exact amount of time is determined by the efficiency of the printer driver, the application, and the application data to be printed. Windows should return control to the user as quickly as possible to remove the perception Windows is slow.

11.4.2. Proposed Solution

11.4.2.1. Overview

To increase system response time by returning control to the application as quickly as possible after user issues a print command. This will be accomplished by spooling metafiles. Since the device driver will no longer be directly involved at print request time, time required for application to issue all output requests should be reduced significantly. Other benefits of this approach are that spooled data is more portable than raw device data, most print jobs will be smaller to reduce network traffic, and conversion of the metafile to device data can now take place on the server vs. on the host. All applications will be able to take advantage of this without any modifications. However, there are some potential issues that may arrise with spooling metafiles (listed below under Outstanding Issues), so in some situations we either won't do it, or will provide both applications & end users with a means to turn it off. The majority of users will wish to take advantage of this. Spooling metafiles will not be supported on pre-Chicago printer drivers since we cannot test to ensure correct behavior on every pre-Chicago driver. If the user has Printman turned OFF, we will not spool metafiles since we will assume the user wants the job physically printed as quickly as possible & does not care about return to app time. In addition, there are cases where the application may know (due to the nature of the data being printed) that spooling a metafile is not desired, so we will provide a way that an application may selectively choose to spool or not spool metafiles. Similarly, we will need to provide an .INI switch to completely disable this for app unaware of this functionality.

11.4.2.2. Affected Components

1) PRINTMAN needs to be modified to recognize Metafile vs. raw output.
2) GDI: Record document information in spool file upon receipt of STARTDOC escape.
3) GDI: Don't delete metafile upon receipt of NEXTBAND escape & return whole page band.
4) GDI: Request text & graphics in same band upon receipt of BANDINFO escape.
5) GDI: In subsequent NEXTBAND or NEWFRAME call use AssignSpoolPage() to assign metafile as spool page.
6) GDI: Modify metafile recording per ViroonT memo.
7) Printer drivers for color devices must support GetDIBits DDI. None do under Win 3.1.

11.4.3. Outstanding Issues

We are aware of several issues (listed below) that arise from spooling metafiles. Current plan to implement a prototype & then investigate these issues & new ones as they arise. Known issues are as follows:

1) Font Mapping: Metafiles only record requests for logical fonts, and have no way of knowing what physical font is actually used. Apps base character placement based on what font they expect to be used & incorrect output results if a different font is substituted. Variation of this is when app installs it's own fonts that get removed from system once app is closed. Same story as above. we know today that PageMaker 4.0 does this.

2) DIB support from printer driver. To support color source blit in metafile, printer driver must support GetDIBits(), currently none do. UNIDRV & Postscript driver will need to add support for GetDIBits(). 3rd party device drivers will need to rev to support this.

3) In some cases (doc w/ lots of bitmaps or src blits), metafile may be larger than raw device data & take longer to create than the actual rendering time.

4) It may be necessary to embed all, or portions of, TrueType font data into the metafile if spooling across the network & a TrueType font is needed by that job & is only available on the host & not the server.

11.4.4. Current Status

11.4.4.1. Overview

All the core work listed under affected components section has been completed as of 8/31/92. Printer driver work & network component work still remain. Testing needs to be done to provide data on known issues listed above & new issues as they arise.

11.4.4.2. Schedule

System

5170	Metafile Spooling Prototype	1	Performance	VirconT	8/5/92	8/19/92
5171	Metafile Spooling Support	1	Performance	LinS	9/1/92	1/19/93

UNIDRV.DLL

Work not yet scheduled.

PSCRIPT.DRV

Work not yet scheduled.

11.5. New Print SubSystem

11.5.1. Objective

The current printing subsystem used by Windows imposes some limitations & has some problems. We need to bypass these limitations & address these problems on a core level, both to improve performance & increase our flexability to accomdate new printing technology as it is developed. Some of the current problems/limits are: Always spooling jobs to disk regardless of available RAM. device contention problems between DOS & Windows applications printing to same port/same time, inability to manipulate contents of spooled jobs. non-optimal use of network resources for printing. We need an interface designed with the intent of support multiple printing clients & multiple printer servers. A key goal here is support the same model in Chicago & NT.

MX 2120604
CONFIDENTIAL

11.5.2. Proposed Solution

11.5.2.1. Overview

We will be using the same API interface as NT to provide consistency. The lower level implementation will be done per the SPIN (Spooled Printing INterface) specification developed by AaronO. Under this implementation, all of the 'work' previously associated with spooling files will be moved from the print manager & tightly integrated within Windows. This approach solves the port contention problems between Windows & DOS Application under Windows 3.1, improves spooling performance by first spooling to RAM & then transparently spilling over to disk only if needed, allow more control of individual jobs in spooled queue, allow a printer driver to be loaded from a network server, and allow a server to convert the metafile to device data. The print manager now becomes a user interface layer to access the printing subsystem. In addtion, these changes allow the RBPA technology developed by WPG to be better integrated into Windows than it is under Windows 3.1.

11.5.2.2. Affected Components

GDI, IFS, Winball components. Full scope of these changes spelled out in \\aaronol\public\spin.doc.

11.5.3. Outstanding Issues

Integrating support for Jumbo/Dumbo/NT servers.

11.5.4. Current Status

11.5.4.1. Overview

Development work underway, current schedule shows code complete by 11/20/92.

11.5.4.2. Schedule

(Note dates on Cougar master schedule have not been filled out yet)

5.18	SPIN. Define IFSMgr like interface for Printing	Networking	AaronO	Done
5.19	SPIN: Implement Spooler VxD	Networking	AaronO	20-Jul-92
5.20	SPIN: Winball redirector modifications	Networking	AaronO	20-Jul-92
5.21	SPIN: Implement PRINT.COM changes to work with Spooler VxD	Networking	AaronO	20-Jul-92
5.22	SPIN: Implement PRINTMAN changes to work with Spooler VxD	Networking	AaronO	20-Jul-92
5.31	SPIN: Provide printing support to Servers	Networking	AaronO	20-Jul-92
5.32	SPIN: GDI changes for printing	Networking	AaronO	20-Jul-92
5.33	SPIN: IFS changes for printing	Networking	AaronO	20-Jul-92
5.35	SPIN: Network provider driver changes for print spooling	Networking	AaronO	20-Jul-92
5.36	Support for NT printing	Networking	AaronO	20-Jul-92

11.6. Mutlitasking Print Manager (32 bit)

11.6.1. Objective

Smooth system performance when spooling print jobs. Increase system responsiveness when canceling a job.

11.6.2. Proposed Solution

11.6.2.1. Overview

.Playback of metafile will be done under pre-emption to preserve system responsiveness. A seperate thread will be spawned to look for cancel requests.

11.6.2.2. Affected Components

Print Manager

11.6.3. Outstanding Issues

User Interface needs to be defined, see section 7.

11.6.4. Current Status

11.6.4.1. Overview

Spec still being designed by shell team.

11.6.4.2. Schedule

Actually a shell work item, listed here for completeness.

11.7. Device Independent Color (DIC)

11.7.1. Objective

Currently, Windows 3.1 does not provide a standardized way for applications to provide direct control over what colors are produced on a given output device, either on screen or on a printer. Applications current generate RGB values to express color which are simply handed to the device driver which will generate color output as it sees fit. No mechanism nor standards currently exist that allow a user to easily obtain printed color output that exactly matches the screen display. This must be addressed if we wish to promote Windows as the platform of choice for Desktop publishing.

11.7.2. Proposed Solution

11.7.2.1. Overview

Briefly, output device drivers (screen & printer) must provide color characterization information to describe how color is reproduced on their device(s) (including limitations of what colors CAN'T accurately be reproduced), some API/DDI calls

need to be added to control/configure this, and a color matching DLL must be provided to map screen & printer color spaces.
A more detailed overview is available in DavidW's "Device Independent Color in Chicago: A Proposal" memo.

11.7.2.2. Affected Components

1) GDI must provide 5 new APIs for apps to gather/set this information
2) GDI must support 4 new DDIs for device drivers to provide this information.
3) Color matching DLL for GDI must be provided by 3rd party.
4) Color characterization format must be defined & exposed to ISV's.

11.7.3. Outstanding Issues

1) Need to decide upon characterization format (HP vs. Apple)
2) Need to sign up 3rd party to provide color matching DLL
3) Need to determine how to expose characterization format for all devices.
4) UI for: how/if to match colors, printer drivers to set color/gamut matching, calibration resetting, color picker?
5) Need to evangelize IHVs to provide drivers w/ color calibration info
6) Determine appropriateness of having UNIDRV provide a default one for color device.

11.7.4. Current Status

11.7.4.1. Overview

DavidW is looking into the outstanding issues listed above, no active development work is planned until 11/92. More details will become available once some the outstanding issues get resolved. Apple is scheduled to announce their color characterization format in 9/92. EFI has also done some work in this area on the Macintosh (ie. Cachet product), we need to speak with them to see if we may able to leverage some of their work.

11.7.4.2. Schedule

5 15.0	Device Independent Color	2	Ease of Use	DavidW	11/10/92	2/23/93

11.8. User Interface changes

11.8.1. Objective

The current printing paradigm is difficult for many users to grasp & understand. There are too many angles to examine in troubleshooting printing problems, making this aspect of Windows difficult to support. WPG has done some interesting studies in the Useablity lab to examine how users view printing, and experimenting with how they respond to different User Interfaces. The work they have done looks promising, and you can see some of the results in the UI of the Jumbo printer driver. For chicago, both the Print manager & printer drivers should provide an improved UI to address these usability issues.

11.8.2. Proposed Solution

While WPG has done some good work in this area, they are interested in positioning their work as a comoditiy & are involved in negotiations to sell this (along with other RBPA technology) to HP. They have not expressed interest in moving this UI to UNIDRV or the Postscript driver. Hence, we need to look at improving the UI for both printer drivers & the print manager ourselves.

11.8.3. Outstanding Issues

While we know the UI needs improving, details need to be ironed out on exactly what form those changes will take.

11.8.4. Current Status

11.8.4.1. Overview

Several people from the shell & printing teams will meet 10/2/92 to discuss printman UI issues, EricBi owns setting up meetings w/ the driver team to define changes for UNIDRV & the Postscript driver.

11.8.4.2. Schedule

Not yet scheduled pending defining work to be done.

11.9. Universal Printer Driver enhancements

11.9.1. Objective

UNIDRV.DLL has made a great impact on improving printer support under Windows 3.1. However, several important print drivers still are not supported under UNIDRV, and some slight deficiencies remain with some devices that are supported by UNIDRV. We need to support these devices under UNIDRV to minimize problems inherent with 3rd party drivers & improve their overall functionality. A detailed list of topics follows below.

11.9.2. Proposed Solution

1) Full support for LJ4 (aka Payette), including HP/GL2 vector mode support. Note that we want to ship this minidriver in Chicago & no longer rely upon HP to provide drivers due to size & quality concerns. HP is currently unaware of this, and has their hands full trying to deliver a LJ4 driver for product introduction. This will be an extreamly sensitive issue for them

since it means they 'lose direct control' of their own driver. We need to handle this situation with care in order to maintain an as positive relationship as possible.

2) Add support for Canon CaPSL (to replace Bob Hoolko's driver from 3.1)
3) Add support for Lexmark's PPDS (to replace Bob Hoolko's driver from 3.1)
4) Remove UNIDRV's screen driver dependencies
5) Various Color improvements in conjunction with DIC system changes.
6) Support new Bi-Directional printers.
7) Add support to support TTY.DRV under UNIDRV
8) Various minor fixes/enhancements to address PSS concerns.

11.9.3. Outstanding Issues

1) All of the development work has been scheduled & is underway. The main issue remaining is determining how we 'break the news' to HP on this issue. In the interest of maintaining a positive relationship, we should do this sooner vs. later.
2) EricBi owns defining work items, waiting on feedback from Canon Europe.
3) Techincal work items have been defined, am waiting to resolve Canon issues due to overlap with Lexmark work.
4) Already sceduled.
5) TBD pending DIC specification completion.
6) TBD as discussed under Section II.
7) Ericbi & Zhanw meet to develop specification after 11/1/92.
8) Mostly already scheduled, need to add "Toshiba fix" to schedule & others TBD.

11.9.4. Current Status

11.9.4.1. Overview

Ericbi working with both Lexmark & Canon to define exactly what work needs to be done & by whom for #2 & #3. #6 still being evaluated.

11.9.4.2. Schedule

4.1.0	Download TrueType font outlines	zhanw		7/15/92	8/7/92
4.1.1	Payette - adapt PCLETTO code for downloading TT outlines	zhanw		8/17/92	8/21/92
	Implement code review changes	zhanw		8/21/92	9/9/92
4.1.2	Payette - split SFDL list into 2: for downloading & for TT font info	zhanw		9/9/92	9/14/92
	Code Review 4.9.4		zhanw	9/17/92	9/22/92
4.1.3	Payette - cache font widths for every point size instead of the last size used	zhanw		9/22/92	9/29/92
4.1.3	Payette - make outline downloading incremental !	zhanw		9/29/92	10/3/92
4.1.4	Payette - integrate downloading TT outlines into HPPCL5MS minidriver	zhanw		10/3/92	10/10/92
4.1.5	Payette - Hppcl5e compatibility	zhanw		10/10/92	10/22/92
4.3.8	GPC - Add PAPERSIZE orientation dependency	zhanw		10/22/92	11/3/92
4.4.0	Color - Enhance DevColor structure	zhanw		11/3/92	11/14/92
4.4.1	Color - improve color printing on HP PaintJet	zhanw		11/14/92	11/21/92
4.5.0	Brush - Use DIBEng or Monochrome Brush	zhanw		11/21/92	11/28/92
4.5.1	Brush - determine if DIB should be use for RGB plane devices		zhanw	11/23/92	1/2/93
4.3.3	GPC - Improve Unidrv/Minidrivers interface (callbacks for adding functionality's)	zhanw		1/2/93	1/9/93
	Bug Fixes	zhanw		1/23/93	2/4/93
4.3.4	GPC - Coordinate DBCS work in Unidrv	zhanw		2/4/93	2/16/93
4.6.1	Size - Look into sharing font metrics between PFM's	zhanw		2/16/93	2/27/93
4.6.2	Capacity - 64K limit on lists for External soft font and cartridge	zhanw		2/27/93	3/11/93
4.3.5	GPC - custom callback for printer dialogs (Kodak proposal)		zhanw	3/11/93	3/16/93
4.3.6	GPC - Add alternative draft mode support for d-m printers (list in DATAHDR)	mikeah		7/15/92	7/18/92
4.2.1	GL/2 - read spec	mikeah		8/5/92	8/13/92
	Code Review 4.1.1	mikeah		8/13/92	8/15/92
4.2.2	GL/2 - Study HPPCL5E driver	mikeah		8/15/92	9/2/92
4.2.3	GL/2 - add basic graphics support (polylines, rectangles, circles,) !	mikeah		9/2/92	9/10/92
4.2.4	GL/2 - investigate/add more difficult graphics support (ellipse, etc.)	mikeah		9/10/92	9/16/92
4.2.5	GL/2 - support line styles !	mikeah		9/16/92	9/28/92
4.2.6	GL/2 - support GL/2 downloading bitmap patterns !	mikeah		9/22/92	10/15/92
4.2.7	GL/2 - support non-orthogonally rotated fonts	mikeah		10/15/92	10/24/92
4.4.2	Color - writing out temp R/G/B files for color plane devices	mikeah/zhanw		10/24/92	10/28/92
4.5.2	Brush - study DIB format and related API/DDI calls ! .	mikeah		10/28/92	11/14/92
4.5.3	Brush - adopt DIB format (Enable, all output calls, dump code, etc.)	mikeah		11/14/92	12/7/92
4.4.3	Color - support pixel color printers using DIB shadow bitmap	mikeah		12/7/92	12/21/92
	Bug Fixes		mikeah	1/4/93	1/30/93
4.4.4	Color - remove the assumption that 'wBlockLen' is less than 64KB & test		mikeah	1/30/93	2/6/93
4.3.1	GPC - Improve device mode dialogs (more possible configurations, etc.)	mikeah		2/6/93	2/20/93
4.5.4	Brush - Implement device-dependent bitmap to DIB conversion	mikeah		2/20/93	2/25/93
4.6.0	Performance - Support new DDI calls	mikeah		2/25/93	3/6/93
4.3.2	GPC - Use SetFormLength Esc for user-defined paper sizes	TBD		n/a	n/a
4.6.3	Size - share PFM's between printer drivers	TBD		n/a	n/a
4.6.4	Add RLE compression	TBD		n/a	n/a
4.6.5	Support grayscale text for TT bitmaps fonts	EricBi		TBD	TBD
4.7.0	Bi-Directional printing - Driver Changes	EricBi		TBD	TBD
4.7.1	Support CaPSL multiple parameter commands	EricBi		TBD	TBD
4.7.2	Download TrueType fonts (raster) on Canon LBP printers	EricBi		TBD	TBD
4.7.3	Download TrueType fonts (raster) on IBM 4019 printer				

11.10.1. Objective

Add limited Postscript level 2 support to address quickly increasing numbers of devices that use this & take advantage of new functionality to achieve performance increases available under Postscript Level 2. Also want to minimize competitive threat from Postscript level 2 driver being developed by Adobe. Improve performance & decrease code size. Allow driver to read "mini" PPD (Postscript Printer Description) files directly to allow easy support for devices not supported in retail product.

11.10.2. Proposed Solution

11.10.2.1. Overview

Support raster data compression & binary encoding of data stream (PS level 2 features) to improve performance. Support enhance color printer information (PS level 2) & integrate into driver per DIC proposal. Use "mini" PPD file format to allow driver to be 100% data driven for printer commands & removes hard coded postscript commands from driver. Other miscelaneous changes to improve performance & decrease code size. Driver will now adhere to ADSC (Adobe Document Structuring conventions) 3.0 specification for PS output.

11.10.2.2. Affected Components

PSCRIPT.DRV

11.10.3. Open Issues

1) Installing a printer driver still currently requires an OEMSETUP.INF file. If we wish to allow PPD files to be used directly, need to determine a solution not requiring INF files. High priority item.
2) If a given PPD file is installed & references fonts other than the "standard" Postscript fonts the driver has hard coded font data for, these "non-standard" fonts won't be available unless the user contacts the OEM & has them provide PFM files. Ideally, the driver should also be able to read AFM files to build PFM files. Low priority item.
3) No support for Bi-Directional communication available if connected serially. Need to investigate if worthwhile to pursue. Expect proposal to be available 1/1/93 in conjunction wth other BiDi work.
4) Several requests to have the driver support "n-up" printing, where the driver prints multiple pages "shrunk" down to a single physical page.
5) EricBi owns investigating Adobe's level 2 driver & make sure we stay competitive against it.
6) Need to modify driver to support downloading TrueType outlines in native mode (vs. Adobe type 1) to non-TrueImage printers. Medium priority, not yet scheduled.
7) Need to significantly improve drivers tracking of printer memory (VM). Windows 3.1 driver has some deficiencies here that are generating several problem reports. Already on schedule.

11.10.4. Current Status

11.10.4.1. Overview

Development work has begun, and we are expecting to receive Adobe's driver by 10/1/92.

11.10.4.2. Schedule

4.9.1	WDP - define the format of WPD 2.0 (and Level II info)	peterwo	7/29/92	8/6/92	
4.9.2	WPD - create enhanced version of [illegible] (PPD parser DLL)	peterwo	8/6/92	8/20/92	
4.9.3	WPD - robustness enhancements	peterwo	8/20/92	8/27/92	
4.9.4	WPD - implement tracking and indexing wpd, ppd and pfm files	peterwo	8/27/92	9/10/92	
4.9.6	WPD - integrate/debug PPD parser (test on all ps printers in Win3.1 and WDL)	peterwo	9/10/92	9/24/92	
	Code Review Changes		9/24/92	9/27/92	
4.10.0	Use WPD - create memory structure and load wpx data	peterwo	9/27/92	10/1/92	
4.10.1	Use WPD - modify driver to use new data structures	peterwo	10/1/92	10/11/92	
4.10.2	Use WPD - replace hard coded strings with PPD strings in dialogs	peterwo	10/11/92	10/18/92	
4.10.3	Use WPD - compatibility bridge from external formats back to devmode	peterwo	10/18/92	10/22/92	
4.10.4	Use WPD - DevInstall() transparently upgrade win.ini etc to new format	peterwo	10/22/92	10/29/92	
4.10.5	Use WPD - integrate/debug	peterwo	10/29/92	11/12/92	
4.10.7	Use WPD - control D off/on support	peterwo	11/13/92	11/15/92	
4.11.0	Support new DDI calls	peterwo	11/15/92	11/22/92	
4.12.1	Robustness - streamline output process/ fdirty bit work	peterwo	11/22/92	11/29/92	
4.12.2	Ease of Use - user selectable automatic paper tray selection	peterwo/davidry	11/29/92	12/2/92	
4.12.3	Ease of Use - papersize for each paper tray.	davidry	12/2/92	12/5/92	
4.13.0	Level II - investigation	peterwo	12/5/92	12/12/92	
4.13.1	Level II - base support	peterwo	12/12/92	12/21/92	
4.13.2	Level II - data compression	peterwo	12/21/92	12/31/92	
	Bug Fixes	peterwo	12/31/92	1/4/93	
4.13.3	Level II - better color support (refer to Adobe's driver)	peterwo	1/18/93	2/1/93	
4.12.4	Robustness - eps printing fixes/enhancements	peterwo	2/1/93	2/15/93	
4.12.5	Robustness - dib and bitblt ROPS	peterwo	2/15/93	3/1/93	
4.15.0	BIDI support	peterwo	3/1/93	3/15/93	
4.12.6	Robustness - Support DSC (better structured output)	peterwo	3/15/93	3/15/93	
4.13.4	Level II - use of global/local VM and better tracking of VM	peterwo	3/15/93	3/15/93	
4.13.5	Level II - binary encoding of print streams instead of using plain ASCII text	peterwo	3/15/93	3/15/93	
4.13.6	Level II - stronger concept of object encapsulation (forms, device config)	peterwo	3/15/93	3/15/93	
4.12.7	Maintain soft font information on a per-printer basis	peterwo	3/15/93	3/15/93	
4.12.8	Adobe - Color separation	TBD	TBD	TBD	transfer functions
	TBD	TBD	TBD		

Adobe - Resolution dependent pattern/hatches halftone patterns	TBD	TBD	TBD
Adobe - Dynamic reconfiguration of printer dialogs, etc	TBD	TBD	TBD

11.11. Minidriver changes

11.11.1. Objective

We want to migrate any printer drivers that were not based on UNIDRV to UNIDRV for chicago. These consist of the Lexmark (IBM) 4019/4029, Canon LBP II, III, & IV, and the generic TTY.DRV. Changes required of UNIDRV are noted elsewhere in this document. In addition, we want to add support for top selling new printers as they are released. We also need to look at establishing some policy for obsoleting drivers for printers no longer in production to minimize our disk space requirements for printer support. We also are looking at creating a Postscript Level 2 minidriver that only supports raster data output & expect that it will offer some significant performance improvements over the current Postscript driver, but at the cost of losing support for device fonts.

11.11.2. Proposed Solution

Negotiations are currently underway with both Lexmark and Canon to identify details necessary to accomplish this. Pending those, we plan to have Lexmark create a minidriver for their products (and font installer), and the MS Ireland Driver development group has committed to creating the Canon minidriver (and font installer). The Ireland group has also committed (& staffed for) to handle all non HP LJ minidriver work for chicago.

11.11.3. Outstanding Issues

1) Lexmark: Technical details have all been outlined as of 9/1/92, some details still outstanding. EricBi owns putting together a detailed proposal of work items required of both companies.
2) Canon: DerryD from the Ireland Driver group has been working with Canon Europa & is still defining the technical requirements.
3) TTY: EricBi & ZhanW need to meet & determine architecture to accomplish this. Meeting not scheduled until after 11/1/92.
4) New drivers: EricBi owns defining policy for how new drivers are added & old ones obsoleted.

11.11.4. Current Status

11.11.4.1. Overview

Outlined under Outstanding issues.

11.11.4.2. Schedule

4.8.1	Create a minidriver for HP LaserJet IV printer	davidry	7/15/92	8/1/92
	Bug fixes for HPPCL5MS driver	davidry	8/1/92	8/12/92
4.8.2	Fix problems in Unitool and update Unitool when spec changes	davidry	8/13/92	12/31/92
4.9.3	WPD - 1.0 (shipped in Win3.1) -> 2.0 converter DLL	davidry	8/12/92	8/26/92
	Code Review 4.9.4	davidry	8/26/92	8/31/92
4.1.5	Payette - HPPCL5E compatibility	davidry	10/2/92	10/9/92
4.3.2	Unitool Update for Papersize changes	davidry	10/9/92	10/16/92
4.2.3	Pscript MiniDrv - using PPD parser	davidry	11/2/92	11/11/92
4.1.1	Pscript MiniDrv - Level II compression	davidry	11/11/92	11/20/92
4.8.4	Evaluate/Write font installer for Canon LBP printers and IBM 4019	EricBi	TBD	TBD
4.8.5	Evaluate creation of minidrivers for Canon LBP printers	EricBi	TBD	TBD
4.8.6	Evaluate creation of minidrivers for IBM 4019 printer	EricBi	TBD	TBD
4.8.7	Evaluate creation of a TTY minidriver	EricBi/ZhanW	11/1/92	11/1/92

11.12. Plotter/Vector Driver issues

11.12.1. Objective

HP has discontinued development support for the HPPLOT.DRV included with Windows 3.1 They are developing a new driver that support HP/GL2 & later plotters, but leaves HP/GL customers without a driver solution. The new driver is also quite large in size (1.1 M uncompressed), not making it an attractive candidate to ship with Chicago.

11.12.2. Proposed Solution

None at this time. NT plans to create a "generic plotter driver" possibly for release 1 to support vector devices, am waiting on information on device characterization format used by it to see if we may be able to leverage off of their work.

11.12.3. Outstanding Issues

see above.

11.12.4. Current Status

11.12.4.1. Overview

EricBi owns following up on this.

11.12.4.2. Schedule

No work items scheduled at this time.

11.13. Windows Printing Group Issues

11.13.1. Objective

WPG is working on several projects that are designed to improve printing under Windows in many ways We need to make sure to address their system requirements as to not create any barriers to implementing the soluuons they are developing, and to assist in those solutions as appropriate.

11.13.2. Proposed Solution

Most of the major concerns voiced against Windows 3.1 have been addressed via the COMM driver changes & SPIN (Spooled Printer INterface) proposal outlined in section IV. All outstanding issues not already covered are described under outstanding issues.

11.13.3. Outstanding Issues

1) Need non-sequential access to pages in print job.(i.e. random access)
Now that we will be spooling metafiles, this becomes a simpler problem: How to add API/DDI support to allow non-sequential access to pages in print job/metafile.
2) Sort order for objects on page not selectable
Objects currently sorted top-to-bottom, which is sub-optimal for some devices.
3) Graphic Object orientation not selectable
Is inefficient when logical orientation is not the same a physical orientation of page.
4) Gradient Fills
Requested from HP.
5) Forms support
Currently we have no support for forms (defined as a macro that can be replayed on many surfaces).
6) No way to clip character
GDI allows app to clip graphics to a character, but this information is not available at the DDI level.
7) Driver should be able to receive point table.
Graphic object may be broken down into trapezoids, with many control points being duplicated. WPG has shown that providing a driver w/ a table of points can significantly reduce printing time.

11.13.4. Current Status

11.13.4.1. Overview

1) Lins investigating as high priority item to possibly add to schedule
2) Drivers can perform this functionality themselves, not scheduled.
3) Too difficult to implement, not scheduled
4) Scheduled to add if resources permit, not otherwise.
5) Lins investigating as med. priority item to possibly add to schedule, need to examine implications under NT.
6) May already be available under NT, Lins investigating.
7) Only of benefit to RBPA devices, not currently scheduled.

11.13.4.2. Schedule

No work items yet scheduled.

11.14. PSS Issues

11.14.1. Objective

PSS has voiced several suggestions on ways in which we could improve printing under Windows, these cover the range from driver changes to help file improvements to system changes. Each suggestion is listed below under the "Proposed Solution" section, with a brief comment as to what we are suggesting to address this.

11.14.2. Proposed Solution

1) Check for a valid "TEMP" DOS environment variable upon startup.
A non-trivial number of the PSS printing problem calls are generated from users have not set this correctly. MSD EXE already checks for this, we should look at having some component of Windows (driver, spooler, user, WIN.COM or ???) check this to cut down PSS calls.
2) Allow driver to provide checkbox for en/disabling :"Fast print direct to port" checkbox This is currently in Control Panel/Printers/Connect dialog. A number of 3.1 customer had to turn this option OFF before they could print correctly. It would be more intuitive for end users to have this option also available from printer drivers dialog (i.e. something else to try in print driver dialog vs. having to go to a rarely used dialog)
3) Better Troubleshooting steps in UNIDRV.HLP/ Better error msg's from UNIDRV.DLL
(i.e. what to try if getting garbage output). Lots of requests for this. no reason why we shouldn't do this. Need to get User Ed to work closely w/ PSS on determining what problems/answers are suggested.
4) Provide dialog listing driver features/functionality/capabilities.

Kind of a vague warm fuzzy at this time. A crude example of this would be the DEVCAPS applets provided by Petzold in his book &/or some of the functionality provide by PRNTTEST.EXE in the 3.1 DDK. Need to work w/ PSS to determine exactly what should go here.

5) Have Driver provide UI for editing win.ini entries.
There were several cases where users needed to add an entry to win.ini for their printer driver. Both users & PSS were often confused on exactly where to add this. Drivers should provide UI for editing these, simple addition to save lots of PSS calls.

6) Have Control Panel/Printers provide option for deleting driver file(s) from disk.
When you remove a printer driver from the control panel, it should provide an option that will either leave/remove the printer driver files from disk.

7) Draft Mode in dot matrix drivers
We 'broke' draft mode for dot-matrix printers under Windows 3.1, we need to fix this in chicago. Already scheduled as a UNIDRV.DLL work item.

8) Provide a "Test Printer" button from printer driver dialog.
Simple to implement, would save PSS techs a lot of time on the phone. Currently when customer calls with a problem where "XYZ app doesn't print correctly", there are a multitude of possibilities (app problem, driver problem, COMM problem) that could be quickly narrowed down with this option.

9) Allow user defined paper sizes for all drivers.
Lots of end user requests for this.

10) Better way of informing users when WDL driver fixes get uploaded.
The actual request was to improve TERMINAL.EXE to have menu option for calling MSDL (Microsoft Download service offered via PSS) to check for new/updated drivers. I think there is a cleaner solution possible, need to research. The real issue is that the printer drivers we shipped in 3.1 had some problems that required us to release fixes on the WDL. We need to improve our testing methods to make sure this situation doesn't happen w/ Chicago.

11.14.3. Outstanding Issues

EricBi owns following up with development on all of these.

11.14.4. Current Status

11.14.4.1. Overview

Waiting on follow-up from ericbi.

11.14.4.2. Schedule

No work items yet scheduled.

12. Applets & Utilities

12.1. Goals:

12.1.1. The eradication of the MSDos prompt

We will provide a complete set of Windows utilities to replace the non-obsoleted functionality provided by MSDOS.

12.1.2. Make Windows users productive

This means we provide things like app setup/deinstall programs. Configuration/ini file management/diagnostic tools. Tools to allow people to take their work between home and office (briefcase).

12.1.3. Make Windows fun, interesting, and sexy

This includes things like system schemes, sounds, multimedia, and perhaps a game.

12.2. Table of Applets/Utilities

Dev Pri	Prog Pri	App	Features	ProgMan Developer	Notes
1	1	WinDisk	Cool disk utilities. Disk Defrag. Disk Fix, Disk Edit, Format Disk, Copy Disk, Sys Disk, FDisk.		
1	1	WinBackup	Back/Restore program. Include tape and network support. Be really cool to allow remote backups.		
1	1	AntiVirus	Virus check a drive. Directory		
1	1	SysEdit	A smart editor to modify autoexec.bat, config.sys, *.ini.Aware of the possible fields that can be in here. Flags potential problems or incorrect data. Tells you if files are missing etc. Is this a control panel applet?	EdH	
1	1	Resource Viewer	Memory/Resource viewer. MemView etc	AaronR	
1	1	Raster	PBrush replacement	EdH	
1		Save/Restore API	Save/Restore of desktop state - Define a mechanism and an api to allow this to happen.	DavidDS	
1		CommDlg	Common Dialog enhancements. File Run with history and browse Exit Windows / Restart Windows / Shutdown Windows. Progress Dialog Box Directory Open (not just file open) MRU list of directories in File Open/Save	ClarkC	
	2	WinHelp	Cleaner UI. Add Full Text search capabilities. Add concept of a HELP directory in the system so that all help files are found here. People can search through all help files at the same time? Maybe make annotations more useful by making them show up at the bottom of the page or at least make them more visible than the paper clip icon; also audio annotations? Add help colors to control panel.		
		WinSleuth	Hardware detection code in windows.		
		WinFix	Windows diagnostic tools. Detect/fix errors in *.ini files, bogus files in the system directories, OLE diagnostices, registry diagnostics, maybe even validates/fixes data files (MMF fixer, WinWord Doc fixer, excel doc fixer etc). Want to validate and fix any cache files we create for Cairo		
	3	Clock	Imbed into desktop. Background faces on clock. Make it part of Cairo Tray. Support syncing to network time source. Use MM Sounds.		
	2	Cardfile	32 bit, toolbars items for searching, deleting, moving items, use OLE II DOC file format so that you can imbed/link into MSMail. Also, want to be able to link mail messages into a cardfile. Also allows the shell to search into a cardfile document. Allow audio recordings in cardfiles.		
		Sparta Apps			What do we do about these. We need UI, our 3d looketc.
		Clipboard	Integrate with Cairo scraps? Make easier to use		Take from spar
	1	Write	OLE server. Mini write. Make work better with text files.		Need to talk wi WinWord group
	1	Packager	Can we drop this now that we are taking OLE 2? Key item is imbeding command line functionality and file links		

		ScreenSavers	maybe AVI screensavers? include sound. Work with DOS boxes. hot spots on desktop to save now and never save.		dos box suppor is done.
		Games	Need action games with sound.		
		CharMap	3D look See fonts in different sizes.		
	1	Terminal	3D look where needed (buttons). Fix bugs. No peek message. No 25 line limit. Wizards? Script language?		Need to talk .. Dynacomm. Wa to use ProCom MicroPhone II?
		Recorder	Should be replace with VBasic? Drop or replace with MSTest.		
		Postit Notes	allow people to put postit notes on their desktop. maybe even on files and other objects in the cairo system. Would be really cool to have a Voice Recorder style post-its.		
		TopDesk	Multidesktop enhancements integrate better with shell/taskman		
		Taskman	Enhance Taskman functionality. Context menu on apps. Taskman on sys menu.		
		BriefCase	Perhaps a drop source on the desktop you simply throw things into so that you can take home. Use astro floppy disk compression schemes to save data to disk. Handle case of multiple disks. Also handle copying of changed files back. Do the easy thing and don't merge portions of changed files and yell about the case of both briefcase file and hard disk file changing.		
		SoundRec	Add enhancedments from Foghorn. Need to add MM Apis to make sound recording easier to implement in apps. Maybe a common dig to do this? Also include compression.		MultiMedia
		Scheduler	Sc..eduler app to allow unattended backups/defrags.		
		VBASIC	Include VB's vbasic.		Can we drop QBASIC?
		MSD	Make it a better PSS tool. Replace Autoexec.bat, config.sys. Stub out win.ini sys.ini with minimal stuff. Include detect code here?		Who is responsible? Setup? P&P?
	1	AppSetup	a generic data driven app install/uninstall program. ISVs need only ship a DLL and an inf file. Support installing apps, fonts, applets, hardware device drivers (.386). Would be really cool if the user simply inserted the floppy, we scanned the disk, and automatically ran the install program giving them options about what they want to install. Need to think about this for network drives/directories. Would also save uninstall data into the user's directory		Owned by setu JohnPa is progr manager.
		Mail/Schedule	Integrate with shell		Who owns? Sparta?
		Chat	Voice Chat. Video Chat?		Multimedia
		MPlayer	Include the new Avi mplayer. Have avi wizards?		Multimedia
		MIDI	Midi enhancements. Keyboard config files. Midi mapping is easier.		Multimedia

HIGHLY CONFIDENTIAL

MX 2120613
CONFIDENTIAL

CMS 00013907

RBC 04308

13. Additional Information

The following sections contain addtional information pertenient to the Chicago spec.

14. Printing Architecture

14.1. Summary

The printing architecture for Chicago must address three issues: performance, great network printing and ease of use. The features which we think we will allow us to reach thease goals are metafile spooling, win32 compatible print subsystem, easy install and configuration, support for bi-di printers.

14.2. Industry Trend

Network printing is the future of corporate printing environment. Whatever we do, we must make sure that our network solution is very robust and easy to use.

We will generally categorize printers as low end printer, i.e. printers which are lower then 300 DPI and high end printers, i.e. printers which have higher resolutions. For low end printers we offer the solution of rasterizing all graphics data on the host and just sending the bits to the printer. For high end printers, we want to define a high level format so that the amount of data we send does not increase as the resolution increases. 300 DPI printers are borderline, but are the most popular today. So we have a mixed solution for them.

14.3. Driver Strategy

Eventually there will be 3 drivers for Windows.

Unidrv - This driver address all the low end printers. It attempts to rasterize everything on the host, with the exception of TrueType fonts for 300 DPI or better devices. We will not ship any non-unidrv based raster printers in Chicago.

PostScript - This driver address all the high end printers which talk in postscript page description language.

Vector driver - This driver does not contain a shadow bitmap. It really works just like the postscript driver. It will take each GDI output call and send it straight to the printer. This driver is intended for devices which match windows DDI calls well including all those winding modes polygons and ROPS. We expect more printers in this category to be introduced in the future. Initially this driver will support plotters with the understanding that it is not a perfect driver for calls which expose the difference between Windows DDI and printer's firmware. This driver will also read the same GPC mini drivers compatible with Unidrv.

In the chicago time frame we will provide the first two drivers. Availability of the vector driver is yet to be determined.

14.4. Spooling Metafiles

Spooling a high level format v.s. raw data increases system response time by returning to the app immediately after the user issues the print command. Rather then having to process all the graphics requests and rasterize the whole page of graphics and/or outline fonts, we simply record the requests in the metafile and return control to the application.

In addition, the size of the spool file will generally be smaller then printer raw data. For a 300 DPI device, printer raw data amounts to about 1MB of data for a full page of graphics. As the industry advances and we start seeing higher resolution devices such as the LJ4 at 600 DPI, it become very expensive to spool raw data.

This feature will benefit all applications - 16 and 32 bit - without modification. Theoretically, new 32 bit applications may create a separate thread to print in the background. With system level metafile spooling support, applications will not have to do this, making it easier to write well behaved apps for Windows.

MX 2120615
CONFIDENTIAL

14.4.1. Why not spool Journal Files

Chicago will spool Metafiles. Spooling Metafiles has a few limitations which we must address. Our alternative is to spool Journal files. Metafiles are collections of GDI calls whereas journal files are collections of DDI calls. NT product 1 will spool Journal files based on 32bit DDI. This approach is not suitable for Chicago because we have a very different DDI. Supporting NT Journal file

CMS 00013909

HIGHLY
CONFIDENTIAL

format will be expensive both in the creation of the Journal file and during the playback of it. Another alternative is to come up with another journal file format based on our current 16 bit DDI, but that will simply create more confusion. We believe that we can identify the limitations of 16 bit metafile and add a few enhancements to work around the problems.

14.4.2. Limitation of Metafiles

The biggest problem with spooling Metafiles is font mapping. Metafiles only record the logical representation of what the application has requested. They do not record what the user is actually getting (physical object). For example, the application may have requested TmsRmn and asked the driver to realize such a font. If such a font is not available, driver will most likely return TNR in place of TmsRmn. Application then makes all the character placement calculations based on the character widths of TNR. During printing, a Metafile with a request for TmsRmn is generated, yet all the character placements are based on TNR because that is what TmsRmn maps to. Later that spooled metafile is given to the print server. Suppose on the server side a TmsRmn font exists then that font will be chosen, the output will end up using TmsRmn font and yet the spacing would have been based on the assumption that TNR was used.
Another font mapping scenario is when the application installs fonts itself. When the user prints, a metafile is spooled requesting special fonts installed by the application. If before the print man has a chance to play out the metafile, the user closes the application causing the application to remove those fonts from the system. The result will be unpredictable as to which font will be used. (We know that PageMaker 4.0 does this).
In order to support color source blit in metafile, the printer driver must support GetDIBits. Otherwise, color information will be lost. Currently no printer driver supports this function. Unidrv and Pscript will be enhanced to support GetDIBits.
If the content of the document contains a lot of bitmaps or source blits, the metafile may be bigger and could even be slower to create then the printer raw data.

14.4.3. PrintMan changes

PrintMan needs to be extended to recognize the Metafile vs. Raw output. When it is a Metafile, a DC is created and the metafile is played back via Banding. NT product 2 should also add this functionality to their product so that it can function as a print server and understand spool files generated from Chicago workstations. Since the reverse configuration (Chicago server and NT workstation) is rare, it is not necessary for NT to spool Chicago style Metafiles.

14.4.4. How is Chicago Different From Rover

In Rover, due to disk space constrain on a Mobile computer, a Pseudo driver will be used. This Pseudo driver maintains a database of printer peripheral information such as paper sizes, printable region, paper sources. This format should be the same GPC format used by mini-drivers in the Universal Printer Driver architecture. The code in the Pseudo driver will essentially be a stripped down Unidrv and uses the same algorithm to interpret the mini-driver data. Rover will not support any Device Fonts and will only support Monochrome output. In order to get full coverage using mini-driver architecture, a postscript mini-driver and a canon mini-driver needs to be written for Rover. These mini-driver do not need data for generating output. They only need peripheral information.

In Chicago, a full driver will be used all the time in order to support device fonts and other private options such as halftoning preferences, color options, user-defined margins and paper sizes.

14.4.5. Compatibility concerns

In order to eliminate compatibility problems, we will assume that metafile spooling does not work with any printer driver until it is tested. GDI will not kick into metafile spooling mode unless the printer driver specifically tells it that it is ok to do so. We will define a new bit in GDIINFO which must be set by printer drivers to indicate that they are compatible with metafile spooling architecture. This bit will be set for all drivers shipped with chicago. This means that third party drivers will need to be upgrade in order to take advantage of this feature. This also gives us a way out if the printer has a different technology making it less efficient to spool metafiles.

14.4.6. What the user will see

There is no explicit way for the user to turn metafile spooling on or off because users should not have to know about spool file formats. Instead, metafile spooling is always on (provided that driver did not veto it) if printman is on. If user turns off printman then metafile spooling is off. The rational is that if printman is on then the user wants control to return to app as soon as possible, but does not care as much about paper drop time. If printman is off, then the user wants the paper printed out ASAP and going through metafile will not be necessary.

14.4.7. What apps will see

We will also allow the application to veto metafile spooling if the application knows that we might hit limitation 3 as described above. For example, if the document contains a large 24 bit per pixel DIB and the output device is monochrome then he might rather take the time hit up front and spool a 1MB spool file rather then a 24 BM metafile for the typical 300 DPI device. Since application has control on a per document basis, the sophisticated applications may pass this decision on to the user at File-Print time.

14.5. Print subsystem

In order to allow windows applications and DOS applications to print simultaneously, we need to introduce a new low level printing interface. In addition we would like to be able to print to an NT server so we decided to adopt the same printing interface as NT. Please refer to Appendix A for all the win32 API's which we plan to support.

14.5.1. The concept of a printer

In NT, there is the concept of a printer. The user is asked to give a friendly name to a printer so that he can use that name to refer to that printer. Internally, a printer is an entity composed of a printer driver, a print processor and a monitor. A printer driver is any windows style printer driver. A print processor is a stream processor. The print processor may take one form of data, process it and produce another form of output. For example, it may take a metafile and convert it to raw print stream. The simplest print processor is one that takes raw data and pump it out to the comm port. A monitor is the piece that watches what goes on at the port. It may display to the user how many percept of the print job has gone by or it may interprete an error coming back from the printer and relay that information to the user. A monitor is optional for defining a printer.

14.5.2. Who will use this print subsystem

Both GDI and printman will use this print subsystem. Rather then having to spool to disk as in windows 3.1 and before, GDI will use the print subsystem to do all the spooling. The print subsystem will take over all the spool job manipulation from PrintMan, reducing PrintMan to be a user interface layer for the print subsystem.

14.5.3. Functions of the print subsystem

The print subsystem will take over most of the spooling functionality implemented by print man today. This print subsystem will offer the following functionality:

- Port contention - This print subsystem will resolve contention for the port from jobs coming from a Windows application, a DOS application or a redirector. All the pieces which access the port must be rerouted through the print subsystem who has sole access to the ports.
- Spooling - This print subsystem will return to its clients immediately by spooling data to memory or to disk.
- Spool Manipulation - The API set includes functions which allow the print jobs to be manipulated. The print jobs can be deleted, paused or rearranged.
- Print Processing - The API set also allows intermediate queue processors to be inserted into the printing stream. Print processors perfrom some trasformaion on a print tream and pass the stream on. For example, GDI may spool metafiles. A print processor which translates the GDI calls to printer raw data must be inserted to interprete the print stream.
- Configuration - The API set will allow print processors and monitors to be loaded and unloaded on the fly.

MX 2120617
CONFIDENTIAL

14.6. New UI

We have a few ideas kicking around about how we want this new UI to look like. Jim Robart in WPG has done some research and wrote up a proposal. In this proposal, printing components are organized as data specific to the Application, the Document, the Print Job, the Spooler and a Printer. Each of these 5 components have an icon,a purpose, a logical association and user-definable attributes associated with it. Please refer to Appendix B for the original proposal.

ChrisG's group will start by looking at NT's print man. We will make a decision as to whether we will start with NT print man or start from scratch. Most likely we will port NT PrintMan since it already interfaces with win 32 print API's. Then we would like to incorporate other ideas we have to it.

A few ideas we have are that we must be able to drag a job from a queue and drop it onto another queue. Given that we are spooling metafiles, the print jobs are less device dependent. As long as a the target device and the metafile were created for devices with the same resolution, there is no problem printing a job to a pool of devices. With a bit more work in enhancing the metafile, we can even print a print job to multiple resolutions and offer the user a choice between cropping or tranforming to fit the page. New UI ideas are also mentioned in stage II of the implementation plan.

14.7. Easy Install and Configuration

14.7.1. Point and click install

Similar to the NT system. We want to be able to just point at a network printer, winball, or NT, and instantly be able to use it without connecting to a port and installing a driver. By writing PrinMan's installation layer to talk to a common win 32 API set we will achieve this with NT systems. We will be able to install and get configuration information from the NT server.

14.7.2. Easy Configuration

We may want to change some of our device mode dialog box to be more graphical, like what the jumbo driver has. We need to find out if users actually like those icons. If this is the case, then it will be worthwhile to incorporate those icons into unidrv and pscript driver. We may be able to store those icons in a central place accessible by both unidrv and pscript so as to reduce driver sizes.

14.8. Bi-Directional Support

Bi-Di printers are essential for delivering a plug-and-play and easy to use system.

14.8.1. Plug and Play

The steps invovled in Plug and Play for any peripheral are:

- Device Detection - to determine the presence of bi-directional perfipherals.
- Protocol negotiation - to transparently negotiate communication protocols.
- Device Identification - to identify perifpherals.
- Once the device is identified, we can load the driver and the associated queue processors. Initially this information may be in the form of win.ini entries. Once the printman identifies that a device of a certain ID is installed, it can look for a particular queue processor and a printer driver to load. In the future, we could upload all the software from the printer.

14.8.2. Status update

Error - For bi-directional printers we will be able to obtain any error during printing and inform the user immediately.

Resource balancing - Resource balacing is this concept of analyzing the CPU power between the host and the printer in order to determine if a a piece of work should be done in the printer or on the host. For example, if the decision of whether a to send out an outline font or a rasterized bitmap will depend on whether the host or the printer has more cpu cycles and memory to spare. I do not expect that we will get into this for printer drivers we develop in chicago. But we should make sure that the architecture we come up with will work well with other groups/companies which may want to use Bi-di to do resource balacing.

14.8.3. Development plan

The implementations of the above mentioned features will be carried out in 3 stages:

14.8.4. Stage I:

Our target for Stage one is to implement metafile spooling and to have win32 compatible print subsystem code complete and component tested.

Spooling metafiles means that when the application requests to print, GDI will record all the application requests in metafiles. Then control is returned to the application immediately. While this is happening the printer driver only answers queries such font, brush, pen realization and character widths. No output calls are seen by the printer drivers. Then the metafile object is sent over to the print manager which then plays the metafile back to a real printer DC as a separate process. This work is currently done and checked into chicago builds.

Spooling support will be provided in the cougar layer and implemented as a VxD. All the services are exposed via the win32 API set. This layer eliminates the contention problem between windows apps and DOS apps trying to print to the same port simultaneously. In addition, since we are sharing the same interface as NT, we will be able to print to printers located on NT servers.

Implement 5 printer driver support functions in GDI by calling into win 32 print subsystem. These API's are OpenJob, StartSpoolPage, WriteSpool, EndSpoolPage, EndJob.

PrintMan will also call win 32 print subsystem when the print job is raw. In the case of metafile spooling, print man will play the metafile back and does not see the print job again.

GDI still spools metafile onto disk rather then going through win 32 print subsystem to do the spooling of the metafile.

14.8.5. Stage II:

In the second stage we will make several enhancements to the spooling format. First let's clarify what is being spooled. For each job there is a job control block that recording the name of the document, the requesting application and the format of the spool job - raw or metafile. If this is a metafile format then there is exatly one metafile per page of output. This allows the printman to offer the user the ability to print any page randomly. For exaple, the user may say, print page 1 - 10 and 20. Or the user may select to print page 15 ten times. If the print job is raw then these options are disabled, sine there is page order dependency when the spool file format is raw due to memory tracking and incremental truetype download.

Enhancements to the current metafile format to make metafile spooling more robust will be designed and implemented. This includes recording the target device's resolution so that a metafile is more device independent. In addition we will record additional information during a SelectFont request to ensure that the same font is realized both at recording time and at play back time.

I need to think a little more what it means to spool metafiles through win 32 print substystem. When this is done PrintMan will merely see a print job coming from the print subsystem. (We no longer use SendMessage to tell PrintMan that a job is done) Then the module which play back the metafile is actually a queue processor. But this is not well thought out.

On the UI side, we should be able to connect to a remote NT server and auto-configure the local print driver. Locally, we should be able to autodect a bi-di printer and configure it. We should also have a simple monitor implemented which allows us to talk to a bi-di printer and update the status such as output of paper or paper jam information.

14.8.6. Stage III:

In the last phase we want to complete device detection work and implement auto-installation of bi-di printers over the network. At this stage we also want to be able to ship metafiles across the network to another chicago print server.

14.9. Win32 PRINTING: REMOTE + LOCAL APIs

14.9.1. Client Printing Interface

BOOL OpenPrinter(pPrinterName, phPrinter, pDefaults)
BOOL ClosePrinter(hPrinter)
BOOL StartDocPrinter(hPrinter, Level, pDocInfo)
BOOL WritePrinter(hPrinter, pBuf, cbBuf, pcWritten)
BOOL ReadPrinter(hPrinter, pBuf, cbBuf, pNoBytesRead)
BOOL StartPagePrinter(hPrinter)
BOOL EndPagePrinter(hPrinter)
BOOL EndDocPrinter(hPrinter)
BOOL AbortPrinter(hPrinter)
BOOL SpoolFile(pPrinterName, portname, lpszJob, pFilename)

14.9.2. Job Control Interface

BOOL EnumJobs(hPrinter, FirstJob, NoJobs, Level, pJob, cbBuf, pcbNeeded, pcReturned)
BOOL GetJob(hPrinter, JobId, Level, pJob, cbBuf, pcbNeeded)
BOOL SetJob(hPrinter, JobId, Level, pJob, Command)
BOOL AddJob(hPrinter, Level, pData, cbBuf, pcbNeeded)
BOOL ScheduleJob(hPrinter, JobId)

14.9.3. Configuration Functions

BOOL EnumPrinters(Type, pName, Level, pPrinterEnum, cbBuf, pcbNeeded, pcReturned)
HANDLE AddPrinter(pServerName, Level, pPrinter)
BOOL DeletePrinter(hPrinter)
BOOL GetPrinter(hPrinter, Level, pPrinter, cbBuf, pcbNeeded)
BOOL SetPrinter(hPrinter, Level, pPrinter, Command)
/* to maintain driver specific configuration information */
DWORD SetPrinterData(hPrinter, pValueName, Type, pData, cbData) DWORD GetPrinterData(hPrinter, pValueName, pType, pData, nSize, pcbNeeded)

BOOL GetPrinterDriverDirectory(pServerName, pEnvironment, Level, pDriverDir, cbBuf, pcbNeeded)
BOOL EnumPrinterDrivers(pName, lpEnvironment, lpDriverInfo, cbBuf, pcbNeeded, lpcReturned)
BOOL AddPrinterDriver(pServerName, Level, pDriverInfo)
BOOL DeletePrinterDriver(pServerName, pEnvironment, pDriverName)

BOOL GetPrintProcessorDirectory(pServerName, pEnvironment, Level, pProcessorDir, cbBuf, pcbNeeded)
BOOL EnumPrintProcessors(pName, pEnvironment, Level, pPrintProcessorInfo, cbBuf, pcbNeeded, pcReturned)
BOOL AddPrintProcessor(pServerName, pEnvironment, pPathName, pPrintProcessorName)
BOOL DeletePrintProcessor(pServerName, pPrintProcessorName)

BOOL EnumMonitors(pName, Level, pMonitorInfo, cbBuf, pcbNeeded, pcReturned)

BOOL ConfigurePort(pName, hWnd, pPortName)
BOOL EnumPorts(pServerName, Level, pPorts, cbBuf, pcbNeeded, pcReturned)
BOOL AddPort(pServerName, hWnd, pMonitorName)
BOOL DeletePort(pServerName, hWnd, pPortName)

14.10. Print UI

Awaiting Jim Robart's UI document.

Since a 32-bit application provides a 32-bit callback function and since the system requires a 16-bit callback function be passed to it, the 32=>16 thunk for the API registering the callback must create a 16=>32 thunk for the callback.

15.1.8. Return Values

In the case of a 16=>32 thunk, it is possible that there will be a loss of precision when converting the 32-bit return value from the API32 to the 16-bit return value returned by the API16. If the return value is an integer (signed or unsigned), it should always be 16-bit significant, so it is harmless to truncate the high word of the API32 return value.

An interesting special case (there are 20 instances in GDI) occurs when the API16 returns a POINT in dx:ax, but the API32 returns the widened POINT in an extra output parameter. Since the 32-bit POINT's members are 16-bit significant in this case, they can be copied from the API16 return value.

15.1.9. User Messages

User messages are transmitted by calls to the message APIs, eg. SendMessage. The message APIs are the most polymorphic of the APIs, with the wMsg parameter determining the semantic meaning of the wParam and lParam parameters. There are 128 different values of wMsg below WM_USER listed in the Win3.1 windows.h.

15.1.10. Register Preservation

32-bit applications expect that the Win32 APIs they call will preserve DS, EBP, EDI, ESI and EBX. The 32=>16 thunks exhibit this behavior.

The rule for 16-bit functions is that they preserve DS, BP, DI and SI. The 16=>32 thunks used in callbacks exhibit this behavior.

15.2. Implementation

15.2.1. Thunk Compiler

The thunk compiler uses a lex-generated lexical scanner and a yacc-generated parser. It converts an input file which defines the relationship between pairs of 16-bit and 32-bit APIs into assembly code. In addition to handling common cases in a general way, the thunk compiler also contains special-case code to handle uncommon cases.

15.2.2. 32-bit versus 16-bit Code

Since 32-bit code does not require segment reloads and since it allows more registers to be used as index registers, it is generally faster than 16-bit code. However, since the helper routines (eg. GlobalAlloc) are 16-bit, the bulk of the thunk code is 16-bit.

15.2.3. Stack Switching

To make stack switching fast, we create a set of overlapping tiled selectors mapping the 32-bit application's stack. Switching stacks in a 32=>16 thunk then just requires finding the best of the previously-created stack selectors. Switching stacks in a 16=>32 thunk (eg. a callback) requires computing the 32-bit stack offset by adding the 16-bit base address and offset of the 16-bit stack. Win32s uses DPMI to get the 16-bit selector base address, but Chicago will use LDT lookup.

15.2.4. >32 Pointer Mapping

The fastest way to perform 32=>16 pointer mapping is by means of LDT lookup. The Kernel provides a read-only LDT selector, into which the 16-bit selector value is an index.

An alternative method is to call through DPMI, but this is much slower.

15.2.5. >16 Pointer Mapping

Since no 16-bit selector may be available which already maps the desired linear address range, a 16=>32 thunk gets a selector from a pool of 16-bit selectors and changes its base address and limit. On return from the thunk, the selectors used by that thunk are returned to the pool for re-use. If the selector pool becomes depleted, it is dynamically grown.

15.2.6. Structure Repacking

The two obvious sources of memory for temporary copies of a structure or array are the stack and the memory manager (local or global heap).

The stack is the preferred source of memory when the object being converted has a small, fixed size. Allocation and cleanup are very fast, but care must be taken not to use up too much stack.

The memory manager is the preferred source of memory when the object being converted is large, or is of variable size (eg. DPtoLP takes a sized array of points as an inout parameter).

15.2.7. Short-term Callbacks

The 32=>16 API thunk writes

```
push <0:32 callback>
push <callback type>
jmp <common callback thunk>
```

on the stack and passes the stack address of this thunklet to the API32. The common callback thunk creates the 32-bit call frame etc. and performs a jmp to the 0:32 callback passed into it by the thunklet.

When the 32-bit callback function returns, it returns to 32-bit cleanup code which restores the stack, registers, etc. and returns to the API16

MX 2120622
CONFIDENTIAL

15.2.8. Long-term Callbacks

15. Thunks

NOTE: These are davidpe's comments in late 1990 about thunking. After we review KevinR's design notes, this section can be deleted.

*** See code in PMWIN that dealt with this stuff.

>>> Do not do hook thunking the way Cruiser did!

- o Have to thunk both directions for 16->32 and 32->16 messaging!
- o Strongly discourage mixed code in a single appl
- o Posted string messages are evil in tiled environment, may not be a problem for WIN32.
- o SendMessage looks at sender and recipient to determine if any thunking of arguments is required.
- o 16-bit apps can Subclass 32-bit controls, so we must support thunking messages.
- o Don't thunk private (>WM_USER) messages, we have no knowledge.

15.1. Concepts

15.1.1. Definition of Thunk

The term "thunk" is used here to refer to the code which converts a call to a 16-bit or 32-bit function to a call of the other type. A 32=>16 thunk converts a 32-bit function (eg. a Win32 API) to a 16-bit function (eg. a Win3.1 API). A 16=>32 thunk (eg. a callback) goes the other way. 32=>16 thunks are the predominant form, used for the APIs and for messages sent from 32-bit windows to 16-bit windows. 16=>32 thunks are used to handle callbacks from an API16 to a 32-bit application-supplied function, as well as for messages from 16-bit windows to 32-bit windows.

15.1.2. Stack Switching

When the application calls an API32 which is really a 32=>16 thunk, the thunk must switch from the 32-bit stack used by the application to a 16-bit stack for use by the system.

15.1.3. Polymorphic Parameters

A parameter which can have more than one semantic meaning is defined to be polymorphic. The most common example in the Windows API set of a polymorphic parameter is one which is a pointer unless the high word is zero, in which case it is an id or an atom (eg. the lpszIcon parameter of LoadIcon).

Other types of polymorphism rely on the value of other parameters. The most common example is in message APIs, eg. SendMessage. Here, the value of wMsg determines the semantic meaning of wParam and lParam. Another example is the hMenu parameter of CreateWindow, which is an HMENU unless the WS_CHILD bit of the dwStyle parameter is set, in which case it is an id.

A polymorphic parameter is dealt with by first determining its classification and then handling as appropriate.

15.1.4. By-value Parameters

Integer (signed or unsigned) parameters which are different sizes on the two sides of a 32=>16 thunk have the high word of the parameter discarded. In the case of a 16=>32 thunk, the parameter is sign-extended or zero-extended, as appropriate.

15.1.5. Pointer Mapping

The 32-bit application code works with 0:32 pointers, but the 16-bit system works with 16:16 pointers. The thunk layer must provide the needed pointer conversions.

Pointers in 16=>32 thunks which only require address translation (eg. strings) are converted by computing the linear base address of the 16-bit selector and adding the 16-bit offset.

Pointers in 32=>16 thunks are harder to handle, since there may not be a 16-bit selector available which already maps the desired region of the linear address space. The referenced data must either be copied to lie within a 16-bit selector's range, or a 16-bit selector must be created/changed as needed.

15.1.6. Structure Repacking

A structure or array which is in a different format in the API32 and the matching API16 must be converted. The conversion must take into account both input and output parameters.

For a 32=>16 thunk, we must allocate memory for the 16-bit version of the object. If the parameter has input semantics, it must be converted to the 16-bit form before calling the API16. If the parameter has output semantics, it must be converted from the 16-bit form after calling the API16. The case for a 16=>32 thunk is similar.

15.1.7. Callbacks

In Win32 compatibility, callbacks are 32-bit application functions which are registered with the system via an API so they can be called from inside the system. One type of callback is the short-term callback, so named because the system only calls the callback during the execution of the API which registers the callback (eg. EnumFonts). The other type of callback is the long-term callback, which can be called after the API which registers the callback has returned (eg. RegisterClass).

Long-term callbacks continue to exist after the return from the API which created them. For example, RegisterClass passes in the address of the WNDPROC callback to be used by all windows of the specified class. The 32=>16 thunk for RegisterClass creates a callback thunk similar to the callback thunk described for short-term callbacks, except that it is in allocated memory, rather than being on the stack. When UnregisterClass is called, the callback thunk is freed.

15.2.9. Code Sharing

At thunk entry or exit, space is saved by sharing common code. The expense is the time taken by one or two near jmp's. As a speed optimization, APIs which need to be as fast as possible can use inline entry and exit code.

Every 32=>16 thunk must switch stacks and make its segment registers 16-bit prior to calling the API16. Since this code is identical for each thunk, space is saved by sharing it. The common code is split between the 16-bit side and the 32-bit side, so two extra near jmp's (one to reach the shared 16-bit code and one to leave the shared 32-bit code) are on the code path for each thunk.

Every 32=>16 thunk cleans up in much the same way (restore the stack and registers, etc.). The only difference between cleanups is the number of parameter bytes popped. Space is saved by sharing cleanup code among all 32-bit APIs with the same number of parameter bytes. The cost in time is an extra near jmp to the common code.

15.3. Special Considerations

15.3.1. Linked-list Parameters

There is only one instance of a linked list parameter, the lpMenuTemplate parameter of LoadMenuIndirect.

16. Edit Controls

16.1. Edit controls and TrueType fonts

This document describes the changes done to the edit control code in USER module inorder to support the TrueType fonts with negative A and C widths.

The edit control code in Win3.1 assumed that the black width of a character does not overlap with the black width of any other character. This is not true with the TrueType fonts with negative A and C widths. As a result, when such a font was used in an edit control and when edit control code drew some characters selectively it inadvertantly erased some portions of other characters on the screen. In some cases, it didn't erase what should be erased. The purpose of the changes done is to remove all these limitations.

One interesting question was whether a negative C or A width of a single character can span across multiple characters. A sample app was written to enumerate all the existing TT fonts, find the biggest negative A and C widths and the smallest advance width for each of them. It was observed that for many existing fonts, a single negative C or A width was manytimes bigger than the minimum advance width, which essentially means that a single character's negative width can effectively span across multiple characters. In other words, when drawing a character, it is NOT sufficient to consider the negative widths of just one character to it's left and one to it's right. Also, in future, somebody could design a very "artistic" TT font that has huge negative widths and very thin advance widths. Our goal is to to modify edit control such that it takes care of the most generic case so that it will work with any fonts that might comeup in future.

16.2. Solved Problems

- We must know where to clip: Because a overhang due to a negative width could erase some other portions, We need to know not only what to draw but also where to clip them.
- When drawing some piece of text, we need to include some characters to the left and some characters to the right because their overhangs could peep into the region that we are drawing.
- Within the same line, different parts of a line may have to be drawn with different color attributes (because of selection). Because of this, we can't draw a line with a single textout call. Some characters may be drawn with some attribute, but their overhangs might fall in some other area where the attribute is different.
- Even a strip of text that has the same attributes throughout, can not be drawn with a single TextOut call if it has tabs in them. Because, Edit control supports user defined Tab settings, we need to take care of the tabs. The space (advance width) generated by a tab could be as small as a pixel. So, it is possible that text on both sides of a tab could overlap with each other.
- When we insert something, we can't draw only the characters to the right of the insertion point, because if there was a huge negative A width overhang, then it could leave some thing to be erased to the left of the insertion point.
- We should have the ability to have margins. Otherwise, the Negative A of the first character and negative C of the last character in every line will get clipped out. But, this margin should not get included in the format rectangle of the edit control; otherwise, it will confuse the end user during horizontal scrolls.

16.3. Implementation Details

16.3.1. Maintaining Negative Width Info:

At the time of WM_SETFONT in an edit control, if it is a TrueType font, we will compute the ABC widths for all the characters and maintain the information in a buffer. This can be done by a single call to GetCharABCWidths() function in GDI. Please note that this function does not rasterize the characters of this font. So, it is not very costly. Moreover, when the first time this function is called, it computes the widths for all the characters in that font irrespective of the number of characters for which we actually asked this function to provide the width information. So, it will be efficient to make this call just once and get the width information for all the characters of a font and store them in a buffer attached to the edit control.

Once we have the information about the ABC widths of individual characters in a font, using this information, we find out the biggest negative A and C widths and the smallest advance width. Using these values we can find out the maximum number of characters that a single negative A or C width can span across. This information is also stored in the edit control data structure. This is the maximum number of characters that we have to consider on either side of a text strip inorder to account for the overhang due to negative A and C widths of TrueType fonts. In the edit control data structure, "wMaxNegA" and "wMaxNegC" are added to store the biggest negative A and C widths for this font as positive values. "wMaxNegAcharPos" and "wMaxNegCcharPos" are added to store the maximum number of character postions that the biggest negative widths can span across. For non-TrueType fonts these values will be zero. So, the same code will work well for non-truetype fonts also.

The "charWidthBuffer" field in the edit control datastructure is a handle to a local memory object. In the case of TrueType fonts, this object will be an array of ABC structures and it will contain the ABC widths of all characters. For non-TrueType fonts, this buffer will just be an array of words and it will contain the advance widths for all the characters. In both the cases, the ASCII value of the character will be the index into the buffer. A new field "fTrueType" has been added to the edit control structure. This field will be used to interpret the contents of "charWidthBuffer".

This buffer is allocated from the local memory. If we don't have enough local mem, then this "charWidthBuffer" will be null. However, we will compute all the information required to handle the negative widths ("wMaxNegA", "wMaxNegC", "wMaxNegAcharPos" and "wMaxNegCcharPos") by either using a temporary global buffer or by calling GetCharABCwidths() for all the characters individually.

It will be inefficient to maintain such a big buffer for Single line edit controls. Also, the amount of text drawing is so little that it may not save us much. So, in the case of single line edit controls, if it is a TrueType font, we always include a fixed number of characters on either side whenever we draw a text strip. The values are defined as MAX_A_WIDTH_OVERLAP and MAX_C_WIDTH_OVERLAP. Currently, these constants are defined to have a value of 5.

16.3.2. Drawing a Block of Text:

MLDrawText(ped, hdc, ichStart, ichEnd) is a function that is the back-bone of multiline edit controls. This function draws all the characters between the character indices ichStart and ichEnd. This function has been re-written to take care of the negative A and C widths.

This breaks down a block of text into lines and each line into one or more strips depending on the selection attributes of the strip. Then it calls ECTabTheTextOut() to actually draw each one of those strips. This function is also re-written to take a clipping rectangle in addition to the usual parameters and this also returns some information about howmuch of this text peeps out of the clipping rectangle due to negative C widths, howmuch of this text must be drawn again inorder to produce that overhang. This negative C width info obtained while drawing one strip is used in deciding where to start drawing the next strip. Extra care is taken to draw the first strip of a line because this could involve a left margin which needs to be blanked out without clipping a negative A overhang; also, care is taken to draw the last strip of a line because the end of a line may need to be blanked out without clipping the negative C width at the end of line. Also, the fact that these blank strips at the begining and end of a line never get drawn with the selection attribute causes some special cases when the first or last text strip of a line has selection attributes.

In the case of single line edit controls, SLDrawText(ped, hdc, ichStart) is the equivalent function. This also breaks the lines into strips based on selection attributes, determines the clip rectangles for them and calls SLDrawLine() for each of them.

16.3.3. Drawing a strip of text:

For multiline edit controls, ECTabTheTextOut() is the function called to draw a strip of text. A strip of text is something where all the characters are to be drawn with the same color attribute. However, we can't use a single TextOut call here because this strip could have TAB characters and the tab lengths have to be taken care of by the edit control code. The tab lengths can be as small as a pixel and hence it is possible that the text portions that lie on either side of a tab can actually overlap each other. So, when we draw those two pieces of text seperately, some clipping will take place. ECTabTheTextOut() draws a text strip opaquely first and in the process finds out those portions of text that get erased because of a overlap. It remembers those portions and then draws those portions transparently. If there are multiple such portions of text within a strip, then all the text between the first such portion and the last such portion are coalesced into a single strip and it is drawn transparently. The idea here is to reduce the length of the text drawn transparently.

For single line edit controls, SLDrawLine() is called to draw a strip of text. This is a much simpler function because this does not have to deal with tabs. However, this has to deal with the case where a password protection character needs to be used instead of the actual characters. This function considers a fixed number of characters to the left and right of a strip when drawing a strip. This avoids all the problems associated with negative A and C widths.

16.3.4. Insertion of text:

When the text to the right of an insertion point has a huge negative A width and peeps into the left of the insertion point, if we insert some text and draw only starting from the insertion point, then we will leave some portions to the left of the insertion point that needs to be erased. This is solved by starting "wNegAcharPos" number of characters to the left of the insertion point.

16.3.5. Margins:

Only the advance widths of characters are considered for the purpose of formatting, word-wrapping and for cursor positioning. We also align the starting positions of the first characters of all the lines. Because of this, the overhangs due to negative A widths of the begining portions of all the lines will get clipped. Similarly, in the case of edit controls with wordwrap, the overhangs due to negative C widths of the last portions of all the lines will also get clipped. These clippings will make the edit controls look ugly. So, we must have a provision for apps to specify a left and right margin that can be used to show these negative A and C overhangs. At the same time, we should not break any of the existing apps. So, we added two new messages that can be used by new windows apps. They are EM_SETMARGINS and EM_GETMARGINS.

EM_SETMARGINS
wParam : Specifies which margins to set. It can be one or both of the following flags:
EM_LEFTMARGIN (0x0001), EM_RIGHTMARGIN(0x0002).
lParam: If EM_LEFTMARGIN flag of wParam is set, LOWORD(lParam) specifies the leftMargin value to be set (in pixels).
If EM_RIGHTMARGIN flag of wParam is set, HIWORD(lParam) specifies the right margin value to be set (in pixels).

NOTES:
1. If the left/right margin value specified is 0xffff, then the "wMaxNegA"/"wMaxNegC" value will be used as the margin value.
2. Once a margin value is set, it remains stored in the "wLeftMargin"/"wRightMargin" field of the edit control datastructure until it is cleared by setting a zero value.
3. When a margin value is set, it modifies the format rectangle of the edit control ("rcFmt") according to the following formula:

```
rcFmt.left  += (NewLeftMarginValue - OldLeftMarginValue);
rcFmt.right -= (NewRightMarginValue - OldRightMarginValue);
```

4. When an edit control is sized, the "rcFmt" is recomputed after subtracting left and right margins from the client rectangle.
5. For backward compatibility reasons, when an app sets a format rectangle using EM_SETRECT, the value given is used as the rcFmt. (i.e) we don't subtract the left and right margins.

The implementation of margins resulted in some problems. When we scrolled horizontally, the left margin portion has to be cleared/redrawn in some cases. In the case of vertical scrolling, the left and right margins are to be included. The code was modified to take care of these cases.

16.4. Potential backward compatibility concerns:

- **The block drawing function in Win3.1 code (MLDrawText(ped, hdc, ichStart, ichEnd)) did not stop the drawing at ichEnd. It always drew till the end of line where ichEnd lies. But, the new function stops exactly at ichEnd. This made some bugs surface and they have been fixed. We don't know whether this will break any apps.**
- The newly added messages to set margins will be used only by newly written apps. So, theoretically this can not break existing apps.

MX 2120627
CONFIDENTIAL

17. OEM Programs Requriements

17.1. Executive Summary

17.1.1. Why OEMs are important for Chicago

"If we design a great product, then the end users will want to buy it, and if the end users want to buy it then the OEMs will want to sell it, and if they want to sell it then they will preinstall it, so why do we have to do anything special? Besides, 99% of the code is the same for the retail and OEM market, so if we build a great retail product, then the OEM can just use it." -- *NOT!*

The OEM is rapidly becoming the predominant sales channel for the Windows product line. Within the next year, it is expected that 70% of the Microsoft Windows sales will be made to our OEMs. As more and more systems are sold with Windows included, and as retail product sales are made to the existing installed base of non-windows machines, the potential market for Retail product sales is decreasing rapidly.

Competitive pressure is increasing in the OEM channel as other operating systems developers realize the strategic importance of preinstalled machines. Recent developments include IBM getting other manufacturers to preinstall OS/2, and rumors of a Novell/DR-DOS version of Windows.

In the Chicago time frame, in order to generate high sales volumes, we are going to have to be quite successful in the OEM preinstalled marketplace. We must make Chicago compelling to the OEM to bundle with their systems. To do this, we must do three things extremely well:

- Make it easier to preinstall Chicago (than the competition) (tools, procedures)
- Make it cost-effective to preinstall Chicago (reduce COGS, time, labor, PSS calls)
- Convince them to build great Chicago Machines (performance, PlugNPlay, architecture)

If the OEM does not have a compelling reason to install Chicago (there are quite a number of truly compelling reasons - dollars, pounds, francs, lira, marks, pesetas, kroner, pesos, ...), or if the competition can provide more compelling reasons, then we will loose sales.

The OEM specific features of the Chicago product are few; for the most part they are features of specifications of other features of the product. This specification can set out the requirements, and the justifications for those requirements, but the actual details must be implemented by other areas of the project.

17.1.2. Basic Chicago Requirements

We must provide tools and techniques to:

Improve Preinstall Utilities for Manufacturing

Improve International support for OEMs

Reduce Cost of Goods

Reduce design-to-manufacture cycle

Improve end-user experience, especially when first turn on computer.

17.1.3. Some Questions to Think About:

What strategy should we take in Chicago to specifically support OEM customers?

What do we need to do to create and propagate the Chicago machine concept?

What kinds of tools can we build to support OEM manufacturing processes, now and in the future?

What are OEM concerns that we need to address?

What are the OEMs goals for preinstalling Chicago?

    Why did they install DOS?
    Why did they install Windows?
    How will they perceive Chicago integeration of the two products?

How do we get more OEMs to install Chicago?

17.2. Specifications

17.2.1. Improve Preinstall Utilities

17.2.1.1. Simplify three phases of use - creating reference system, installing on manufacturing line, and end user experience
Priority: 1
Owner: Setup

17.2.1.2. Improve flexibility to handle different manufacturing scenarios, from disk duplicators to network installations (eg: modular setup architecture)
Priority: 1
Owner: Setup

17.2.1.3. Allow OEM-callable Setup segments (a setup api?) - OEM customization and promotional screens, factory Quality Control tests, control panel segments, apps copy/installation for build-to-order systems
Priority: 2
Owner: Setup

17.2.1.4. Allow OEM selection of drivers to remain on hard disk after post-install
Priority: 1
Owner: Setup

17.2.1.5. Verify (checksum, etc) validity of copied files during/after preinstallation
Priority: 2
Owner: Setup

17.2.1.6. Allow multiple reboots before end-user w/o affecting post-install
Priority: 1
Owner: Setup

17.2.1.7. Install permanent Swapfile during manufacturing or post-install
Priority: 1
Owner: Setup

17.2.1.8. Allow OEM to configure group window sizes, locations, contents
Priority: 2
Owner: Setup

17.2.1.9. Support preinstallation and configuration of multiple application software packages
Priority: 3
Owner: Setup

17.2.1.10. All features should be optional - controlled by .ini or .inf flags
Priority: 1
Owner: Setup

17.2.1.11. Option to over-ride automatic hardware detection during preinstall if not preinstalling on reference machine that is different from target machine
Priority: 1

Owner: Setup

17.2.1.12. Option to provide specific set of choices for printers, etc. when in post-installer
Priority: 1
Owner: Setup or Control Panel?

17.2.1.13. Use .CFG files in post-install that were copied to target machine, but not present during reference system setup.
Priority: 1
Owner: Setup & Plug-N-Play

17.2.1.14. Preinstall/OEM Install/Postinstall options should be modules within the standard setup, not separate OEM versions.
Priority: 2
Owner: Setup

17.2.2. Improve International Support for Preinstallation

17.2.2.1. Provide preinstall functionality in all localized versions
Priority: 1
Owner: Setup

17.2.2.2. Provide integrated, but optional, multilingual options
Priority: 1
Owner: Setup

17.2.2.3. Reduce disk space required for localizaton modules (use resources)
Priority: 2
Owner: International

17.2.2.4. Speed up multilingual conversions for end-user (all files should use resources)
Priority: 2
Owner: International

17.2.2.5. Multilingual choice should be Windows/GUI - preinstalled system should not need character mode
Priority: 2
Owner: Setup

17.2.3. Reduce OEM and MS Cost of Goods

17.2.3.1. Minimize printed documentation
Priority: 1
Owner: User Ed

17.2.3.2. Minimize floppy disk count
Priority: 1
Owner: All

17.2.3.3. Minimize manufacturing time (eg: copy files)
Priority: 2

Owner: Setup

17.2.3.4. Minimize manufacturing labor (eg: auto-configure)
Priority: 1
Owner: Setup and PNP

17.2.3.5. Reduce PSS Support calls (both for setup and long term)
Priority: 1
Owner: Setup & All

17.2.3.6. Provide some sort of floppy generation or backup utility to create system diskettes. May need to keep directory of special, unmodified Windows files to generate rather than diskimage.
Priority: 1
Owner: Setup & Utilities

<u>17.2.4. Reduce OEM design-to-manufacture cycle (time to market)</u>

17.2.4.1. Provide Hardware Compatibility Tests
Priority: 1
Owner: PSS

17.2.4.2. Provide good, early information on Chicago Machine Design
Priority: 1
Owner: Evangelism group, Plug-N-Play, OEM

17.2.4.3. Minimize number of different "Reference" versions of Windows OEMs maintain by improving hardware detection and/or automatic configuration in Post-Installer
Priority: 2
Owner: Setup & Plug-N-Play

<u>17.2.5. Improve end-user experience, especially when first turn on computer.</u>

17.2.5.1. Simplify post-installation
Priority: 1
Owner: Setup

17.2.5.2. Dialogs and Wizards for registration (generally improve registration & licensing)
Priority: 1
Owner: Setup

17.2.5.3. Registration may include an encoded serial number that the user enters from documentation.
Priority: 3
Owner: Setup

17.2.5.4. Allow re-registration if system transferred/sold to new user
Priority: 2
Owner: Setup

17.2.5.5. Dialogs and Wizards for non-autoconfigure questions in Setup
Priority 1
Owner: Setup

17.2.5.6. No Fragmentation Problems with new system after post-install
Priority: 1
Owner: Setup

17.2.5.7. Performance tuned for best response on first power-up by user
Priority: 1
Owner: Setup & All

17.2.5.8. Easy to Use Initial Menu system, intro to Windows (MS or OEM provided)
Priority: 2
Owner: Apps group and Setup

17.2.5.9. Install Windows tutorial and demos as icons in an Introduction to Windows or Start Here group.
Priority: 1
Owner: Setup

18. International

This appendix groups the Intl features that can be found throughout the Chicago Specification.

18.1. Overview

This part is a high-level overview of the problems we are trying to solve with Chicago, of the areas that need attention and of the solutions we are proposing.

18.1.1. Localization:

Our number one problem is quality of the localized software and timeliness. The question we are trying to solve for Chicago is: "How to produce a high quality localized operating system (i.e. which compares to the same level of quality/functionality's as its English equivalent) and how to release this localized operating system at the same time as the US version ?".

The problems facing quality and timeliness today for Windows and mostly MS-DOS are associated with the way we build localized software and the impact it has on the localization process and testing. For all MS-DOS components and some Windows ones, each localization requires first a lengthy and painful process of translating strings in .asm files, and second a complete re-compilation to produce a localized version of the same element. This localization and build process requires that we re-test all the OS base functionality to make sure that our build/localization process did not introduce a bug during the painful translation process. With all the languages we are chartered to do, this process can prevent us from meeting our timeliness goal. On the other hand, if we skip localized version functionality testing and try to meet our time constraint, we are almost certain that our localized OS will not meet our quality standards and we are risking a release disaster.

The only way we can achieve fast and high quality software localization is to make all localizable elements "no-compile". By no-compile, we mean that any localizable strings, messages, dialog boxes must be easily accessible. For example, where possible, we should use our Windows resource model. If not possible, we should have all messages in separate files that would be loaded at run-time. Dealing with strings this way helps make the translation and updating process fast and safe. It does not require any compilation of the product therefore minimizing the impact on testing.

Note: one of the side-benefit of no-compile can also be found in our multi-lingual pre-installation process. Today we rely on inefficient patching utilities for all our MS-DOS files.If all our elements were no-compile, this process could be enhanced and be more flexible

18.1.2. NLS support:

The way NLS (National Language Support) is being done today in Windows and MS-DOS is inconsistent. Both systems also lack a lot of functionality's that application would require from an operating system to support National Languages better. Finally, MS-DOS NLS support UI has been a nightmare ever since it existed (it was created by IBM).

The question we need to solve in Chicago is :"How to make Windows and MS-DOS (a.k.a. Chicago) NLS support consistent for both the DOS and Windows world ?". The other important question is "What additional NLS functionality's are required by DOS and Windows Application to make Chicago a versatile Intl. product ? ".
Our first task in this area will be to synchronize Windows and MS-DOS NLS functionality. Everything we can do at Windows level must be reflected at DOS level. For example, the user must have one consistent way of changing the keyboard layout from Windows and these changes must be reflected transparently at DOS level (unlike the situation today where any changes made in Windows are completely ignored by DOS, resulting in a hybrid environment).
Our second task is to make Chicago a great Intl. platform for applications. Today's Windows Intl. support for application is limited to some win.ini settings accessible through "GetProfileString" type functions. Applications have been looking for better/easier support such as Intl. API's. etc.
Our Third and final task is to look at the usability of our current International interface in Windows and MS-DOS and make it more intuitive and more flexible.

CMS 00013928

18.1.3. Multi-Lingual support:

HIGHLY CONFIDENTIAL

MX 2120634
CONFIDENTIAL

RBC 04329

Today, Windows and MS-DOS are mono-lingual systems. But since the day we introduced common components (Win31) and allowed applications to replace system components, we introduced the notion of a multi-lingual operating system.

The issue Chicago must address has two aspects. First a problem to solve: "how to authorize applications to replace system components without introducing language incompatibilities (for example, having French Winword run under US Windows and keep the file-open dialog in French"). The other aspect of this issue is a potential area of improvement and of competitiveness (that neither OS/2 nor the Mac have today): "how could we make it possible for Chicago to be a true multi-lingual system where users could just install additional languages and switch them on the fly ?".

The first question addresses a design flaw in Windows 3.1 and its solution can be treated as a postponed bug fix. The second question calls for a more broad solution and together with our localization improvement opens the door to additional lines of products and additional revenues for Chicago.

18.1.4. Unicode support

It has been decided that Unicode will not be part of the US Chicago product. However, there are still open issues regarding its implementation for our Far East, Middle Eastern and Eastern Europe editon.

18.2. Solutions and Feature Sets:

This part contains a set of feature requests for addressing the problems mentioned above

Important Note: The features presented below are affecting the whole product and most of its components. They can be found individually in the main body of the Chicago Specification.

For each feature, you will find the following information:

- A title of the proposed feature.
- A development priority, that should be understood as follows:
  - *Priority 1*: Must do
  - *Priority 2*: Should be scheduled
  - *Priority 3*: Should be considered
- *Affected Area:* Area of the product where development will be needed.
- *Description:* A description of the proposed feature.
- *Work items:* Identification of the work items associated with the feature.

18.2.1. Localization support

The solution to our localization problem is clearly to go to a 100% no-compile scheme for all localizable elements in Chicago. This means that translating strings and including them into the binary files should not require any re-compilation.

18.2.1.1. Fix localization problems with SETUP.INF and .INI files

Priority: 1
Affected Area: Setup
Description:
Currently, SETUP.INF (and all .ini or .inf files) has become so complex that it is almost impossible for localizers to distinguish the information to be localized from the ones that must be left alone. This often results in fatal localization errors that will severely impact the installed software. We propose to redesign the .inf structure so that all information to be localized is clearly identified or isolated in a separate include file.

MX 2120635
CONFIDENTIAL

18.2.1.2. Add a LoadStringHandle API

HIGHLY CONFIDENTIAL

CMS 00013929

RBC 04330

Priority 1
Affected Area: User, Applets
Description:
The LoadStringHandle API achieves the same functionality as the LoadString API, that is, to read strings from the resources. The difference between them is that LoadStringHandle allocates the text buffer on the local heap whereas LoadString requires the application to allocate the buffer. We have seen in the past many bugs with localized version of our products. The application developer allocated text buffers that were too small for foreign languages, and the international tester failed to discover the bugs because they don't understand all the languages. With the new scheme this class of bugs should disappear.

18.2.1.3. Store localizable objects as resources in all Windows components

Priority: 1
Affected Area: All Windows components
Description:
All localizable elements must be stored as resources for all the Windows files. This is to allow no-compile localization.

Work items:
The files that need to get converted to using resources (instead of Hard Coded strings) include:
- Setup (Dos part)
- Display drivers

18.2.1.4. Store localizable objects as resources in all Cougar components

Priority: 1
Affected Area: All Cougar components
Description: All localizable elements must be stored as resources for all the Cougar files. This is to allow no-compile localization. An acceptable alternative for pieces that cannot use resources (like Kernel) is to store all localizable elements in a separate message file, dynamically loaded at run-time or store all localizable elements in the .exe header (resource-like format).
Note that this msg file (format etc) structure has been partially implemented for some of the Cougar pieces and there is existing code to pull strings from the file. We still need link this code into the various vxds, dos apps, kernel.

Work items:
The files that need to get converted to using resources (instead of Hard Coded strings) include:
- Kernel
- All VxD's
- All Dos Utilities
- Winoldap.mod (?)

18.2.2. NLS Support

The solutions to our problems lay in three major areas. We must first synchronize the NLS support in Windows and MS-DOS and create a consistent approach for the user and the application to access International information from Windows for the whole system. There are then additional International features that we want to consider to enhance our Application International support. Finally, we want to review the current Windows and MS-DOS NLS User Interface.

18.2.2.1. Code Page synchronization for DOS VM and Windows

Priority: 1
Affected Area: Cougar
Description:
VM's must be at all time synchronized with Windows regarding their OEM Code Page. The Code Page used for the file system is set once and for all at setup time. But if a user changes Code Page in a VM, his Clipboard Transfers and Fonts should reflect the current VM's code page and translate to Windows character set.
We will not add a Windows UI to this as it is a DOS VM only function. The new SetNls (see below) command line should be the only way to change code page in a VM.

Work items (Mikecr ?)

18.2.2.2. NLS synchronization for DOS VM and Windows

Priority. 1

Affected Area. Cougar, Control Panel.

Description:

VM's must use the same default NLS information (date format, currency, etc.) as Windows system default. When the default is selected at Windows level, VM's must be synchronized and use the same country information.

This feature must include a Windows UI. In fact, there should be one single UI that set NLS defaults in Windows and in a VM at the same time.

Work items (Mikedr/Shell team ?)

18.2.2.3. Expand international information in Custom Setup and add country/keyboard detection to "Upgrade" Setup (Express and Custom).

Priority: 1

Affected Area: Setup

Description:

We must add a "Sub-Language" field to the custom setup field. This should allow the user to customize his default country information at setup time. The combination of language/sub-language will allow setup to identify countries in a unique way. NT uses this exact same method. If a user decides to change the default for this field (which is set for a particular language version, i.e. US/English has US, French has French, etc.), then Setup needs to write the new country values to WIN.INI [Intl] section and set the default "locale" for the NLS API's (if implemented).

In addition, our Upgrade Setup should detect which Country/keyboard was installed in the previous version (by checking config.sys and autoexec.bat). The new system should have the exact same settings as the old ones, especially if we want to emphasize "Express" setup.

Work items: Marcw/Johnpa

18.2.2.4. Add NLS API

Priority: 2

Affected Area: New Component (.dll), Win32s, 32bit only

Description:

The NLS API is an interface for retrieving and processing "locale" specific information. A "locale" is a combination of a language/sub language. The goal is to provide an interface that allows the programmers to easily write locale-independent code.

The API set is broken into two areas, locale-dependent string transformations and locale manipulations. The dependent locale string transformations are: Uppercasing, lowercasing, collating and mapping strings. Locale manipulation includes handling locales and getting information about locales like currency, date format, date strings and fifty others.

This scheme retrieves locale information on a per API basis (and so on a per application basis). It is moving away from the current scheme where NLS information are system wide for just one language. It will allow two applications dealing with different languages to run at the same time. It also adds a real API level Intl. support for Application.

Work Items and estimate:

Port dll from NT to Chicago (2 weeks)

18.2.2.5. Add Control Panel support to NLS API's and VM NLS

MX 2120637
CONFIDENTIAL

Priority 2

Affected Area: Control Panel

Description:

We need a new User Interface to support the new NLS API's, the VM NLS information and the old Windows .ini settings. This interface seems more likely to be in the Control Panel.

Work Estimate (Shell team)
About 2 weeks.

18.2.2.6. Use NLS API's in applets

Priority: 2
Affected Area: Applets
Description:
All our applets must use 100% of our NLS API's to obtain country-dependent information. This includes using all country defaults from the Control Panel, using the string comparison routines to perform sorting, etc. This is especially important as these applets are supposed to be models for applications.

Work Items:
Support of NLS API's must be added to the following applets if they are 32 bit. If not, they should use the old .ini settings:
- Control Panel
- Pifedit
- Mail/Schedule +
- Terminal/Recorder
- All new applets.

Time estimate: this should be included in the 32bit conversion if any.

18.2.2.7. Finish Cougar SetNls feature: This is the simplified DOS command line NLS control.

Priority: 2
Affected Area: Cougar
Description:
We need to give the DOS user an easy way to change NLS parameters from the command line. Today's command line does not include any simple interface to change settings such as Country information, etc. This feature should bring under one simple command line all the current Intl. parameters that can be changed for NLS support at DOS (Country, Code Page, Keyboard).
Note that some of the work in this area has already been done and is on stand-by. A specific spec is available.

Work Estimate (Mikedr ?) Note that the work has been partially done.

18.2.2.8. support multiple keyboard layout

Priority: 2
Affected Area: User, Control Panel, Cougar
Description:
We should provide the means for users in Europe, Canada, the Middle East and other regions of the world to quickly switch between multiple keyboard layouts. This feature is planned for Windows NT/Cairo.
Our assumption will be that at all time we will have one keyboard layout default for the whole system. If a user change the layout through our new API's/UI, then this keyboard layout becomes the new default for the system. Also, we assume the VM's would follow the same model as they will be synchronized.

Work Items:
- Modify the keyboard driver to support multiple dll's
- Add an API to Get/Set keyboard, plus add the concept of keyboard ID's
- Modify the Control Panel applet to specify the sets of keyboard layouts that need to be pre-loaded. Also modify Control Panel to specify the hot key combination to use to switch layout.
- Add UI to show the current state of the layout
- VM's synchronization (Mikedr)

Time Estimate: ?.

18.2.3.1. Add Language ID's to Windows resources to allow for multi-lingual resources in executables

Priority: 3

Affected Area:

Description:

We want to be able to include more than one language into executable resources. This is to allow us to fix the multiple language common component problems (see below) and it also gives us the potential of making Windows multi-lingual. This would allow a user to add extra language to their already installed version and it would let them run the system in different languages if it was needed.

To do this, Windows must be able to include multiple resources in its executable, and tag them with language ID's. We must also provide a mechanism to display a particular language when chosen.

Note that NT has this model already in place.

Work Items:

This format is already defined and in place for Win32 executables (from NT). The key work items here would be:

1. Implement this resource format for in the Win32S kernel
2. Modify USER to use the new apis and work with the new resource formats where applicable.

This would be a feature only for 32 bit apps.

18.2.3.2. Add Language ID's to Cougar resources to allow for multi-lingual resources in executables

Priority: 3

Affected Area: Cougar

Description:

We want to be able to include more than one language into executable resources. This is to allow us to fix the multiple language common component problems (see below) and it also gives us the potential of making Cougar multi-lingual. This would allow a user to add extra language to their already installed version and it would let them run the system in different languages if it was needed.

To do this, Cougar must be able to include multiple resources in its executable, and tag them with language ID's. We must also provide a mechanism to display a particular language when chosen.

Note that NT has this model already in place.

Work Items: Some of the work has supposedly been implemented.

18.2.3.3. Add multiple language resources to common components

Priority: 2

Affected Area:

Description:

Based on the multiple resource model (see above), we must allow applications to add extra language resources to the common components if they are installed on an existing system that runs a different language. This feature should allow applications to add an extra resource in the existing component and to retrieve a specified language resource at run-time. The key existing component is common dialogs and new ones will appear as the shell is further refined. The main problem here is this feature is for 32 bit apps and common DLLs is a 16 bit component. However, the decision has not been made as to how the common components would be implemented (16bit or 32bit).

Work Items: (Davidds)

18.2.4. Other Items

18.2.4.1. Provide complete international support in SDK

Priority: 2

19. Documentation

19.1. Overview

This document provides a quick overview of user ed (UE) planning for Chicago so far.

The Chicago UE team will focus on making Chicago fun to learn and easy to use. We're planning the following components:

- A tour/video that quickly introduces new users to the Chicago features.
- Wizards that actually perform difficult or complex tasks for the user; for example, installing hardware or troubleshooting a problem.
- Concise printed documentation focusing on the tasks that users really need to do. The documentation will be highly visual in order to effectively link the interface with the users' tasks.
- Help that provides context-sensitive information about the interface and about how to accomplish tasks. We're thinking about including bubble Help, dialog box Help, demos or short tutorials launched from Help topics, and Help topics launched from error messages.
- Error messages that provide more extensive user assistance in the form of Help, Wizards, or additional troubleshooting aid.

The following sections describe the UE plans and recommendations more completely.

19.2. Tour/Video

Marketing research shows that users, especially high-volume customers, like to have an introduction as a first step toward learning the product. From a tour/video, the user gains a base from which to explore the product features. In addition, a tour/video gives us a chance to show how the operating system enables other products.

Users want a product that is easy to digest and compelling in its own way. We discussed two possibilities for fulfilling this customer request: a self-running guided tour in the product and a stand-alone video.

19.2.1. Issues

We need to do additional research before we can make a solid recommendation. The following are some of the issues we discussed:

- How can the tour/video have lasting value? Building marketing objectives into the tour/video would help extend its usefulness. We could make it possible to launch parts of a tour from online Help wherever it makes sense.
- How can we move the user from the tour/video into the docs (print or online)? Some successful videos make references to page numbers or sections of the documentation. Showing a user in the tour/video looking up procedures in the User's Guide or finding information in online Help are possibilities.
- What kind of budget would be reasonable? Both videos and tours carry their own types of resource requirements and overhead. Videos take time and money. Tours take time and disk space.
- Would we be able to efficiently localize the tour/video? This idea needs research. Videos can be dubbed, and ease of localization can be a feature of the tour.
- We need some way to familiarize the upgrade user with the new interface quickly. It's possible that a tour/video could be a learning tool for our current Windows users.
- A tour/video is a passive way to remove intimidation, especially if it includes a person, in a familiar scenario, using Windows to successfully get work done.
- A tour/video would be an opportunity to leverage one effort spanning marketing, sales, and user ed

19.2.2. Recommendation

Research the feasibility. The following is a list of some things to learn:

- What group at Microsoft has done a successful tour/video?
- What did it take?
- What user feedback have they had?
- What does marketing think?
- What's our recommendation for content?
- What are the costs?
- Can the tour/video replace the current Windows tutorial or does the tutorial serve a separate purpose and need to be revised for Chicago?

19.2.3. Wizards

Wizards assist people in performing tasks. A Wizard walks a user through a procedure by displaying the appropriate dialog boxes and performing the task after the user specifies all the necessary information. In Chicago, we'll include Wizards for some of the more complicated or common tasks.

We came up with the following possibilities for Wizards:

- Setup--Help with network installation and general Setup issues, troubleshooting Setup
  Plug and Play hardware (although if it's really plug and play, why would you need a Wizard?)
- Troubleshooting
- OLE
- Configuration tasks (Control Panel tasks: streamlining AUTOEXEC.BAT and CONFIG.SYS)

- Optimizing
- Printers (setting up and sharing)
- Dialing

Wizards are resource-intensive; they require resources from development and testing as well as user ed and program management. We think that in some cases, it may be just as easy to redesign the interface as to create a Wizard to perform a task.

19.2.4. Recommendations

We need to find out more about the process for creating wizards and then brainstorm other tasks that need Wizards. Before we can decide which Wizards to create, we must find out about the new user interface in Chicago. Then we'll have to cut our list of possible Wizards down to a manageable amount.

19.3. Printed Documentation

We will produce a core two-volume set: a Getting Started manual (working title) and a User's Guide. The goal is to produce concise, modular documentation that can be easily adapted to different retail SKUs. The core books serve as the printed documentation for the full Chicago product (Windows with integrated networking and MS-DOS) and form the basis, with some modification, for Windows without networking and a stand-alone MS-DOS version. In addition, we will create a Quick Reference card that can be adapted for each product. It will provide fast access to task-oriented information, such as key combinations and mouse tasks.

The printed documentation is not comprehensive, but rather, provides information on the essential tasks that users perform. Detailed technical information for support technicians and administrators is not included in the retail documentation package. This material is successfully presented in a Resource Kit.

19.3.1. Getting Started

The Getting Started book presents first-time use information, such as how to set up, what the system requirements are, what's new and different, and focused troubleshooting assistance aimed at getting people up and running (other troubleshooting information will be presented online). Page estimate is 100 pages, or less if most of the troubleshooting information can be presented online.

19.3.2. User's Guide

The User's Guide is task-oriented, very visual piece, that succintly presents the essential information users need to accomplish work. In the past, Windows documentation has been organized by application (for example, File Manager and Print Manager). The documentation followed the organization of tasks in the interface. Eliminating the distinctions between applications in the Chicago shell design makes writing truly task-oriented documentation easier. The aim is to get information about the most essential tasks (as identified through marketing and user reseach) presented in clear language that does not rely on jargon. Extensive use of screen art, callouts to interface elements, and clear procedural information will link the interface to the written book and aid users in becoming oriented and productive in this new environment. We'll schedule adequate time to ensure that the printed documentation is extremely well-indexed. Page estimate is 200-250 pages.

19.3.3. Quick Reference Cards

Quick Reference cards are included in many documentation packages but have not been used in Windows or MS-DOS documentation sets. Cards might be a bit costly, but they provide a lot of information in a portable format. As was noted in a Cairo user ed survey, many people use Quick Reference cards until they learn the product. Possible Quick Reference cards include the following.

- MS-DOS command summary
- Top 10 user tasks, with cool graphical layout
- Introductory Windows material

We should explore options for providing the electronic version of the documentation on CD-ROM. More and more, large sites that license Windows want text that can be used with a Viewer and full-text search.

Documentation will be produced by using Word for Windows and the Microsoft Standard Design. Close coordination with localization will ensure that the international and domestic product can ship at the same time.

19.3.4. Issues

- Will COGs support three separate pieces in retail package?
- Can highly visual documentation be effectively localized now, given the new Microsoft Standard Design and a standard set of tools?
- What material is most effectively presented in print?
- How can we ensure that printed documentation works with online information?
- For Quick Reference cards, what's the cost, with and without color and graphics? Who uses them? How popular are they? What type of material is best presented in them?

19.3.5. Recommendations

- Provide concise, highly visual User's Guide.
- Put first-time use topics into a short Getting Started book.
- Include a task-oriented Quick Reference card in the retail package.
- Put more technical, support-related information into a Resource Kit rather than in retail documentation.

- For large accounts, explore ways to provide books on CD-ROM.

19.4. Online Help

Previous Help designs were modeled after the Windows user interface, with each application having its own Help file. As we move away from this application-oriented model, the Help model will change accordingly. Until we have a clearer idea of how far Chicago deviates from Windows 3.1 interface, it is difficult to give any details about the infrastructure of the Help system. We do know that the following types of Help are being considered.

Balloon (or bubble) Help—Useful to the beginner user who is unfamiliar with the interface. It would be useful if the user could remove this Help at any time.

Context-sensitive Help—Includes information about tasks, dialog boxes, and possibly troubleshooting, as well as teaching skills needed to work with the interface. We are also investigating how effective Help is when launching short tutorials from individual Help topics.

Online books—Could be an effective way of presenting certain conceptual information that is not included in print.

Regardless of which types of Help we use, the Help-system design will work together with the online tutorials, demos, tours, and wizards, as well as with the printed documentation.

19.4.1. Issues

Online help is a critical part of the Chicago documentation set. As the size of our printed manuals shrinks, more importance is placed on the online Help. In order to insure that Help meets this challenge, we must include certain things in the overall Chicago design.

Most important, program management must commit to a fixed amount of disk space for online materials. This includes Help, demos, tutorials, and tours. It is very difficult to plan a Help system and then alter it at the end of a project. Also, the amount of disk space reserved for Help dictates the type of graphics, if any, that can be used, and the type of cross-referencing we can include.

The second consideration is the type of search and indexing capabilities our authoring tool provides. An effective and efficient search engine and index are critical. Users must have a high level of confidence that they know what information is online and that they can find what they are looking for.

Finally, we should usability test our current help to determine if our approach in presenting task-oriented and context-sensitive help is effective. A much higher burden for learning and using the product is placed on help now that the page counts for printed documentation has been reduced.

19.4.2. Recommendations

- Research the effectiveness and benefit of bubble help
- Usability test current Windows and MS-DOS Help to determine how effective it is in presenting information. Focus on how users find information in the Help.
- Look at competitors' Help systems and models to determine how effective these are in presenting information

19.5. Error Messages

Error messages are probably one of the most important components of any software product and often do not get the attention they deserve. Depending on how specific, detailed, and well-written error messages are, users either will be able to effectively troubleshoot their problems or will end up feeling frustrated.

An excellent article entitled "Error Messages Don't Get the Respect They Deserve" was recently published in one of the trade journals. This article states that "Error messages are the negelected step-children of most products. They often look like the last thing the developers considered, and that's a shame." For Chicago, we want to address this problem.

19.5.1. Issues

- When should the planning, writing, and coding of error messages take place?
- What type of error messages are most effective, both for end users and for Product Support Services?
- What type of user assistance can we provide with error messages?

19.5.2. Recommendations

Error messages should be as specific as possible, documenting only one, or at the most, two, error conditions. Substituting predefined MS-DOS or network error codes should be avoided when possible, since these messages are often too general or vague to be of any real use to the end user. In the past, Product Support has requested that each error message have a unique identifier, such as a number, in order to help technicians browse through the Knowledge Base and troubleshoot problems over the phone.

Error messages should include some form of user assistance. Depending on the error condition, some messages might include effective troubleshooting techniques within the message itself. More complex error messages that may require the user to follow several steps to solve a problem should include more elaborate forms of user assistance, such as context-sensitive Help, or possibly even Wizards

'or error messages to be as informative and helpful as possible, we agreed that error checking needs to be designed into the code early on in the design and development phase. Instead of being an afterthought that occurs when the code is being implemented, error conditions should be well thought out while the code is being designed. During the design phase, developers and designers should consider all the possible error conditions that can occur and then work with a writer to develop the most descriptive and useful error message for each condition. As the code is implemented and tested, bugs will undoubtedly occur, requiring additional error checking. Development and program management should keep user ed in the loop and solicit their assistance in writing additional error messages and in providing additional user assistance, if needed.

19.5.3. Issues

As the Chicago project settles into a clearer feature set and user interface, our content issues will be resolved. The research we do over the next several months will also help to solidify our plan. We are still left with a number of concerns, as follows:

- What's the best way to balance printed documentation and online material?
- Do users want printed documentation at all? How much?
- How do users work with online Help?
- What about a user interface makes it easy to learn?
- What is our disk-space budget?
- What is the COGS goal?
- Will we be able to adequately staff a team to do Wizards? (headcount)
- How committed is Chicago leadership to provide ease of learning?
- Why isn't a UE person included on the UI team?
- Who does the Resource Kit?
- How much can PSG modify the winhelp engine?
- What will the MS-DOS 7 product look like? Will it be a subset of Chicago?
- What's the relationship between Chicago with and without networking? What will be the implications for the interface? Will it be possible to use the same documentation and share Help files?
- Can we influence the terminology choices being made by the OLE team?
- What research resources are available to us?
- What authoring tools will we use for online authoring?
- Should the Windows 3.1 tutorial be revised for Chicago?
- What CBT tools will we use?
- How will our plan impact localization?

19.6. What's Next

We will have some time while we wait to see a Chicago interface. We should use the time we have up front to get involved in planning and design meetings, to test previous documentation decisions, and to research how people learn to use an operating system. The following list summarizes what we'll make happen next.

- Include user ed in Chicago planning and design meetings. Work with program management to define the product focus and refine the user education strategy.
- Test documentation decisions we have made in the past. We'd like to know whether users learn by using our printed tour. We'd also like to know more about if and how people use our context-sensitive Help in dialog boxes to learn about performing tasks.
- Work with marketing to hold focus groups that help us find out how users learn to use an operating system.
- Work with marketing to visit local users. We can learn about the tasks that people perform when using an operating system and how they learned to do them.